University of Wolverhampton
School of Computing and IT

An Investigation into Quality Assurance of the Open Source Software Development Model

Tobias Otte, Dipl.-Wirt.-Inf. (FH), MSc.

A thesis submitted in partial fulfilment of the
requirements of the University of Wolverhampton
for the degree of Doctor of Philosophy

June 2010

Signature……

Date ……

Abstract

The Open Source Software Development (OSSD) model has launched products in rapid succession and with high quality, without following traditional quality practices of accepted software development models (Raymond 1999). Some OSSD projects challenge established quality assurance approaches, claiming to be successful through partially contrary techniques of standard software development. However, empirical studies of quality assurance practices for Open Source Software (OSS) are rare (Glass 2001). Therefore, further research is required to evaluate the quality assurance processes and methods within the OSSD model.

The aim of this research is to improve the understanding of quality assurance practices under the OSSD model. The OSSD model is characterised by a collaborative, distributed development approach with public communication, free participation, free entry to the project for newcomers and unlimited access to the source code. The research examines applied quality assurance practices from a process view rather than from a product view.

The research follows ideographic and nomothetic methodologies and adopts an anti-positivist epistemological approach. An empirical research of applied quality assurance practices in OSS projects is conducted through the literature research. The survey research method is used to gain empirical evidence about applied practices. The findings are used to validate the theoretical knowledge and to obtain further expertise about practical approaches.

The findings contribute to the development of a quality assurance framework for standard OSSD approaches. The result is an appropriate quality model with metrics that the requirements of the OSSD support. An ideographic approach with case studies is used to extend the body of knowledge and to assess the feasibility and applicability of the quality assurance framework.

In conclusion, this study provides further understanding of the applied quality assurance processes under the OSSD model and shows how a quality assurance framework can support the development processes with guidelines and measurements.

Acknowledgements

This thesis could not have been completed without the continuous encouragement and support of many individuals and organisations. I acknowledge them most sincerely and I would like to express my deepest gratitude to all of them.

Especially, I would like to thank my supervisors Prof. Robert Moreton and Prof. Dr. Heinz-Dieter Knoell for their helpful guidance, invaluable advice and their time and effort spent in reviewing this thesis. I would like to thank Prof. Dr. Heinz-Dieter Knöll for his useful support in the early stage of the thesis and his encouragement to facilitate lectures at the Leuphana University of Lueneburg. I wish to express my grateful appreciation to the principal supervisor Prof. Robert Moreton for his constructive criticism, his constant patience and guidance and his support in obtaining funding from University of Wolverhampton to enable the attendance at conferences during this research project.

I would like to thank all colleagues form the Leuphana University of Lueneburg (Institute FFB) for guiding me during the statistical analysis and Dr. Mylatz for providing the resources needed to complete the work. I gratefully acknowledge their important advice and help.

I would like to thank all OSS developers who participated in the pilot study, the main survey and the interviews that were conducted within this research. Especially, I would like to thank all participants of the case studies for their willingness to support this project, their time and effort, their constructive feedback and helpful ideas. They contributed largely to the successful completion of the research.

Finally, I wish to express my deepest thanks to my family for their encouragement and support throughout. Especially, I would like to express my sincere gratefulness to my beloved wife Christiane. She has always believed in me and has shown an endless understanding of my weekend and part-time work. Her patience, encouragement and support largely made this thesis project possible.

List of Original Papers

This thesis includes the following original papers published in the proceedings of international conferences and journals:

I. KNOELL, H.D., OTTE, T. and MORETON, R. (2008) Quality Assurance Methods and the Open Source Development Model. *FINAL Information Science: Technical Report and Working Papers,* **18(1)**, Leuphana University, Lueneburg, pp. 95–106.

II. OTTE, T., MORETON, R. and KNOELL, H.D. (2008a) Applied Quality Assurance Methods under the Open Source Development Model. *in Proceedings 32nd Annual IEEE International Computer Software and Application Conference 2008, 28.7.-1.8.2008, Turku, Finland,* pp. 1247–1252.

III. OTTE, T., MORETON, R. and KNOELL, H.D. (2008b) Development of a Quality Assurance Framework for the Open Source Development Model. *in Proceedings Annual IEEE 3rd International Conference on Software Engineering Advances ICSEA 2008, October 26-31, 2008, Sliema, Malta,* pp. 123–131.

The author of the thesis is the principal author of the above stated papers, which are referenced in the text.

Contents

List of Figures

List of Tables

Abbreviations

BRR	Business Readiness Rating
CMM	Capability Maturity Model
CMMI	Capability Maturity Model Integration
DoD	Department of Defense
EDOS	Environment for the Development and Distribution of Open Source Software
FSF	Free Software Foundation
GPL	GNU General Public License
GQM	Goal Question Metric
IEC	International Electrotechnical Commission
IS	Information System
ISO	International Organization for Standards
KPA	Key Process Areas
LGPL	GNU Lesser General Public License
MPL	Mozilla Public License
OS	Open Source
OSQ	Open Source Quality
QAfOSS	Quality Assurance Framework for Open Source Software
OSS	Open Source Software
OSSD	Open Source Software Development
OSMM	Open Source Maturity Model
PCS	Process Capability Score
PCS_MCW	Process Capability Score percentage model coverage weighted
PSS	Project Success Score
PSS_PW	Project Success Score pre-weighted
PSS_W	Project Success Score weighted (individual)
PSS_EST	Project Success Score estimated

PSS_SEL	Project Success Score weighted selected
PDCA	Plan Do Check Act
QA	Quality Assurance
QIP	Quality Improvement Paradigm
SEI	Software Engineering Institute
SPICE	Software Process Improvement and Capability Determination
SQA	Software Quality Assurance
SQP	Software Quality Program
SVN	Subversion
SW-CMM	Software Capability Maturity Model
TQM	Total Quality Management

Part 1 - Quality Assurance for Open Source Software: Background

The first part of the thesis provides a background about the research area. The research aims and objectives are examined and an overview of the research methods is presented. An appropriate research method is chosen to represent the underlying structure of the thesis.

In the literature review, certain criteria for software quality, quality assurance and measurement methods are discussed. The different quality models, grouped by product, process or improvement approaches are evaluated.

In addition an introduction to Open Source Software, the underlying development model and the collaboration approach is presented. The advantages and shortcomings of the Open Source Software Development model are discussed with regard to software quality aspects. The necessity for further evaluation of quality assurance methods is demonstrated.

1 Introduction

Open Source Software (OSS) is developed by freely participating programmers, who distribute source code in a collaborative, virtual and geographically distributed environment, communicating over the Internet (Raymond 1999; Feller and Fitzgerald, 2000, p.62). Software projects are classified as Open Source by the licence under which the software is distributed, which enables a remarkable development approach. In contrast to proprietary software development, which is classified by plan, schedules, resources and deliverables, the OSS model starts with an idea, followed by a more prototypical approach with frequent release cycles. This development process gains momentum due to the availability of the code and the participants' motivation (Feller and Fitzgerald, 2000, p.62).

The OSSD model delivers successful products that seem to be high quality, such as Linux or the Apache Web Server. It uses unconventional methods such as unrestricted access to the source code, large user involvement, voluntary developers, less design or planning and no task assignment (Mockus *et al.*, 2002). The Open Source Software Development (OSSD) model differs from traditional plan-driven approaches. While traditional methods have defined teams and requirements, the OSSD follows an iterative and parallel development approach with a user driven development direction, no central management, free participation, large development communities and effective user testing. Users are considered as co-developers, release cycles are more frequent and defect handling occurs as collaboratively within the whole community.

In recent years, the focus of the OSSD model has changed due to increasing economic interests, which resulting in a higher need for reliability and sustainability (Michlmayr 2007). Nowadays, OSS products are introduced by millions of users, relying on the development approach, which requires higher importance is given to software quality due to growing commercial use. This questions the effectiveness of quality assurance processes under the OSSD. In particular, larger projects may need more organized approaches to manage complex development activities. Furthermore, the distributed development approach faces certain challenges, such as communication and coordination (MacCormack *et al.*, 2001; Rasters 2004).

There is evidence that the OSSD delivers high quality (Halloran and Scherlis 2002; Aberdour 2007) but diverse practices and methods are applied to achieve this goal. Advocates of the OSSD claim the high quality comes from largely making use of peer reviews, user testing and debugging in parallel. One key element may be structured processes and best practices which suggest what methods to apply with what effect. Furthermore environmental factors, organisational structures or management skills become important aspects. However,

empirical evidence of quality assurance methods for OSS are still lacking (Glass 2001). In recent years, QA activities under the OSSD model have been subject to some investigation, such as by Zhao and Elbaum (2000, 2003), Halloran and Scherlis (2002), Koru and Tian (2004) and Michlmayr (2005, 2007). This research contributes to the understanding of QA practices, but the question of how to achieve constantly high quality in voluntary distributed OSS projects needs to be solved. In particular key processes which contribute to high software quality under the OSSD model need to be defined. These questions gain a higher importance with the growing commercial interests in OSS and focus on the degree of reliance of the OSSD model.

It can be argued that the OSSD model may require the definition of a quality standard, which supports the entire lifecycle (Stamelos *et al.*, 2002). Such a standard may offer guidelines or checklists on how to conduct quality assurance processes in the OSSD model. However, a common QA model under OSSD is absent. Further research is required to understand how quality assurance practices within an OSS model are applied and what processes are required to achieve these goals. An analysis of applied practices and methods may deliver generalisable results that could contribute to a more general model.

1.1 Research Question

The research aim is to investigate the key processes that assure software quality under the OSSD model. The research examines the process characteristics and explores how these findings contribute to the development of a QA framework. Furthermore, it evaluates the concept and assesses how an improvement of the development approach can be made.

Thus, the central questions of this study are posed as follows:

- *How is Software Quality Assurance in OSS projects applied and what key practices characterise mature projects?*
- *Can a quality assurance framework contribute to the improvement of the quality assurance activities in the OSSD?*

These research questions constitute the underlying structure of the thesis. Subsequent research questions result from the research aims and objectives as described in the following.

1.2 Aims and Objectives of the Study

The research contributes to the development of a QA framework. The framework needs to reflect OSSD model characteristics and necessary QA methods. A critical analysis of the

existing theory and further research is undertaken to gain empirical results about applied QA methods and practices. A QA framework is developed and critically examined in practical OSS projects. The findings contribute to the improvement of software quality assurance methods under the OSSD model.

The objectives of this research are to increase the understanding of quality assurance practices and to investigate common processes and methods that support the OSSD process.

- The literature review provides the definitions of quality assurance and discusses quality models. The OSSD model is introduced and an overview of recent research in the subject area is presented.
- An investigation of the OSSD with regard to weaknesses of quality assurance and improvement potential is undertaken. Quality criteria for OSS are evaluated by surveying OSS projects. The survey research method is used to gain evidence about the applied quality assurance methods under the OSSD.
- The empirical findings are analysed with regard to applied quality approaches in mature projects. These results confirm existing theories and provide further knowledge of applied key practices.
- The results contribute to the development of a QA framework, which describes QA processes and methods to assure the product quality. A measurement model is considered to assess the process quality and to suggest product quality metrics.
- The research explores how a QA framework can affect and support the process and product quality in OSS projects. The case study method is considered to extend the body of knowledge and to assess the feasibility and practicality of the framework.

1.3 Research Contribution

The research contributes to literature and to knowledge about applied QA methods under the OSSD model. Furthermore, empirical evidence about QA practices is provided.

The findings extend the research in Software Quality Assurance by introducing a framework approach for the OSSD model. The QA model proposes a novel process framework. Moreover, the model offers a tailorable approach supporting OSS projects and might set a novel quality assurance standard within this area.

In addition, the findings may contribute to other software development models, following a collaborative, distributed software development approach. The framework may be adapted for further improvements.

This research may be used as basis for further research within this subject area. Further studies could explore the applicability of the quality assurance framework in longitudinal studies.

1.4 Summary of Research Approach and Methodology

The domain "Information Systems" (IS) is multi-disciplinary as "contributions to the study of information systems come from the natural sciences, mathematics and engineering, from the behavioural sciences and linguistics" (Land 1992). Choosing the appropriate research methodology is itself an area of debate, as discussed in the technical correspondence in the Communication of the ACM (Galliers and Land 1988; Jarvenpaa 1988). Galliers (1992) recommends a range of approaches for the use in the general field of IS. The available approaches to researchers fall broadly into the categories of scientific and interpretivist (Galliers 1992). As computer science has its foundations in the empirical sciences, this positivist approach assumes that observations of the phenomena under investigation can be made objectively and rigorously. The characteristics of scientific research are described by repeatability, reductionism and refutability (Galliers 1992). The strengths and shortcomings of these characteristics, when applied to IS research, have been discussed in the literature by Checkland (1981), and Galliers (1992). Hirschheim (1985) argues that IS draw heavily from social science because IS are fundamentally social, rather than technical systems. "Thus, the scientific paradigm adopted by the natural sciences is appropriate to information systems only insofar as it is appropriate for social science" (Hirschheim, 1985, p.9). The underlying assumption for this research methodology in IS needs to ensure that "qualitative research conducted as science should complement non-qualitative research" (Kirk and Miller 1986). In order to choose the appropriate research method for this thesis, it is important that the methodology ensures the research aims can be satisfied and that the process is academically sound. The research model of Galliers and Land in Jarvenpaa (1988) reflects an appropriate methodology for IS research, suggesting a "chain of methodologies":

- Research Question
- Survey Research
- Theory Building
- Case Study
- Theory Testing in a Field
- Theory Extension

This research model is selected and largely reflects the structure of the thesis. The research uses ideographic and nomothetic methods and follows an anti-positivism epistemology. The refinement of the developed model is achieved by using qualitative methods with case studies following an interpretative approach. The analysis of social data is required, rather than statistical evaluation. The latter is inappropriate for the applicability of field studies methods.

Therefore, a widescale "Theory Testing in a Field" is not applied within this research. A detailed discussion of the selected research methodology and assumptions is provided in chapter 4.

1.5 Thesis Structure

The thesis structure follows the research methodology as outlined by Galliers and Land (1998) and is divided into three parts. Figure 1 outlines the main structure of the research in conjunction with the research method.

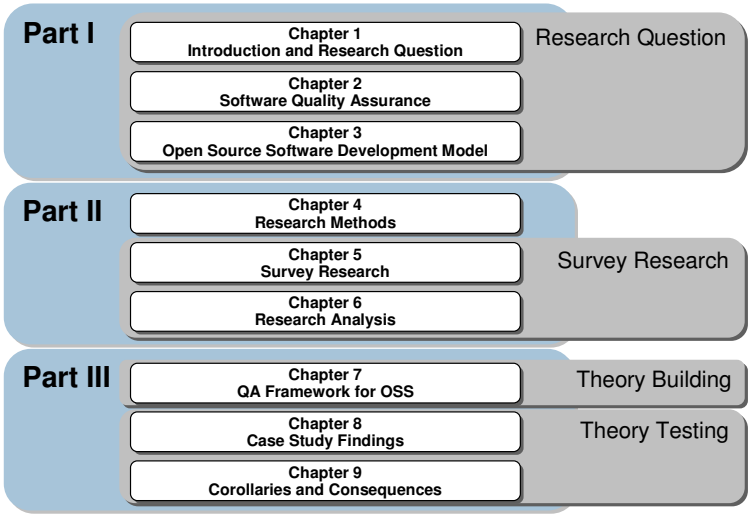

Figure 1. Thesis Structure

Within the first part, the problem statement is examined and literature is discussed. In chapter 1, an introduction to the topic is presented and the research problem explained. In chapter 2, a literature review is conducted on the topic of the research. The definitions of software quality as well as important quality models are evaluated. In chapter 3, an introduction to the OS concept is given and the recent findings of quality assurance activities under the OSSD model are explored.

The second part discusses the underlying research methodology and focuses on the first research question, investigating applied QA practices under the OSSD. Chapter 4 discusses the epistemological and ontological assumptions and the appropriate research method. Nomothetic and ideographic research approaches are used to gather and analyse data in the study. In chapter 5, the survey research approach is described. The development of the questionnaire and the data collection method are explained. Chapter 6 focuses on the analysis and discussion of the survey results in order to complete the findings from literature.

The third part concentrates on the development of a framework and its practical verification. Chapter 7 explores the taxonomy of a framework. The key elements that contribute to Soft-

ware Quality Assurance (SQA) under the OSSD are collated in a framework and a measurements model is developed. In chapter 8, the case study approach and the findings are discussed to verify the model components and to prove its contribution to the OSSD model. The final chapter summarises the key findings and their contribution to the knowledge base. An overview is presented and directions for further research are suggested.

2 Quality Assurance: An Introduction

This chapter will provide an overview of Software Quality Assurance (SQA). Definitions of the terms Software Quality, Quality Assurance and Metrics are provided and the different quality models are discussed.

2.1 Software Quality

2.1.1 Definition of Software

The IEEE 610 (1990, p.66) Software Engineering Terminology Standard defines software as "Computer programs, procedures and possibly associated documentation and data pertaining to the operation of a computer system". The term "possibly" in the IEEE definition shows that some factions believe that documentation and data need to be included but others not. In this thesis, software is defined as a generic term for organized collections of computer data and instructions that can be broken into two major categories of system software - the basic non-task-specific functions of the computer and the application software, which is used to accomplish specific tasks.

2.1.2 Definition of Quality

> In a perfect world, everybody would see the same thing when they think of 'quality'; and they'd see the same thing before the contract was signed and after it was delivered. (Cameron Low in SQM, 1993)

According to Hoyer and Hoyer (2001), two major camps can be identified when discussing the definition software quality - "conformance to specification" and "meeting the customer needs". The definition "conformance to specification" states that the quality of products and services can be measured with characteristics that satisfy a pre-defined fixed specification. In the definition "meeting the customer needs", quality is defined independent of any measurable characteristics as the capability to meet customer expectations. Quality is difficult to define and means different things to different people. Juran (1979, p.2) states "quality is fitness for use". On the one hand Juran focuses on the use of a product, which corresponds with customer expectations and requirements; on the other hand, he underlines the importance of quality of design and conformance.

David Garvin (1984) concludes that quality is "a complex and multifaceted concept" and considers five different perspectives on how quality can be viewed. These views have their source in areas such as philosophy, economics, marketing and operation management:

- The transcendental view
- The user view

- The manufacturing view
- The product view and
- The value-based view

The transcendental view sees quality as something that is not defined, although the meaning is obvious (Pirsig 1974). The user view describes quality as fitness for purpose: "Quality consists of the capacity to satisfy wants" (Edwards, 1968, p.37). This user view is evaluative and examines the product in terms of meeting user requirements. The manufacturing view comes along with the definition “conformance to requirements” (Crosby, 1979, p.15), which considers the process of construction and assesses to what degree the product is developed "right" the first time. The product view sees quality associated with the characteristics of a product and adds value by changing its features. Abbott (1955, pp.126-127) defines this quality view as the "differences in quality amount to differences in the quality of some desired ingredient or attribute." In the value-based view quality refers to the price the customer is willing to pay (Kitchenham and Pfleeger 1996). Feigenbaum (1991, p.1) describes this view as "quality means best for certain customer conditions. These conditions are (a) the actual use and (b) the selling price of the product." The view considers the trade off between cost and quality, which is reflected in the ISO 8402 definition.

The international ISO 8402 standard defines quality as “all the features that allow a product to satisfy stated or implied needs at an affordable cost".

2.1.3 Definition of Software Quality

The ISO 8402-1986 provides an extended outline of what the attribute software quality should be and defines software quality as “the totality of features and characteristics of a product or service that bear on its ability to satisfy stated or implied needs”. This definition is used in the remainder of the thesis.

2.2 Software Quality Assurance

Juran (1979, p.2) defines quality assurance “as the activity of providing to all concerned the evidence needed to establish confidence that the quality function is being performed adequately”.

The IEEE Standard Glossary of Software Engineering Terminology defines Quality Assurance as:

- “A planned and systematic pattern of all actions necessary to provide adequate confidence that the item or product conforms to established technical requirements” (IEEE 730, 1998, p.3).

- "All the planned and systematic activities implemented within the quality system, and demonstrated as needed, to provide adequate confidence that an entity will fulfil requirements for quality" (ISO/IEC 12207, 2007, p.6).

Schulmeyer and McManus (1999) conclude that because of the placement of these definitions in the software-related glossary these definitions also apply for Software Quality Assurance (SQA). Furthermore, Schulmeyer and McManus (1999) cite Baker and Fisher (1982, p.105) who focus on organisational aspects of responsibility for performance and defines "SQA as the functional entity performing software quality assessment and measurement". According to Fairley (1997) software quality assurance involves both process and product assurance. From the organisational perspective, SQA is commonly seen as management of software quality programs. Dunn (1990, p.11) argues that "SQA does not assure the quality of software, but the effectiveness of a software quality program." The ISO 12207 (ISO/IEC 12207) describes SQA as a process providing adequate assurance that the software product and development process fulfil their quality requirements. Therefore, if software quality is built into the product through the use of a process that has quality built in, SQA must be an aspect of all software development activities (Dunn 1990). According to Ralston and Reilly (1993, p.226), SQA must be applied throughout the development process. They argue that the focus of assuring software quality should emphasise more the process than the product. For this reason, the definition by Schulmeyer and McManus is considered as the underlying definition of SQA within this thesis:

> "SQA is the set of systematic activities providing evidence of the ability of the software process to produce a software product that is fit to use." (Schulmeyer and McManus, 1999, p.9)

In this thesis, the term SQA is used for quality assurance activities for software development.

2.3 Software Quality Measurement

The IEEE 1061 (1998, p.2) describes measurement as "the act or process of assigning a number or category to an entity to describe an attribute of that entity." Fenton and Pfleeger (1997, p.5) state, "measurement is the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to characterize them according to clearly defined rules." Quality measurement is expressed in terms of metrics. First of all definitions of the terms metrics, measures and indicators are provided and the relation between them is explained.

According to Schulmeyer and McManus (1999, p.403) a measure is "to ascertain or appraise by comparing to a standard". A standard or a unit of measurements is e.g. dimension or capacity, which is used for something determined by measuring. Without an expected value to

compare against a measure does not provide useful information (Schulmeyer and McManus 1999).

A metric is a quantitative measure of a degree to which a system, component, or process possesses a given attribute, that is a calculated or composite indicator based upon two or more measures (Schulmeyer and McManus, 1999, p.403). Perry (1991, p.542) classifies the metrics into three categories; anomaly-detecting, predictive and acceptance.

- *Anomaly-detecting* metrics identify deficiencies in documentation or source code. These deficiencies usually are corrected to improve the quality of the software product.
- *Predictive metrics* are measurements of the logic of the design and implementation. These measurements are concerned with attributes such as form, structure, density, and complexity.
- *Acceptance metrics* are measurements that are applied to the end product to assess the final compliance with requirements. Tests are an acceptance-type measurement.

An indicator is "a device or variable that can be set to a prescribed state based on the results of a process or the occurrence of a specified condition" (Schulmeyer and McManus, 1999, p.403). An indicator provides a quick comparison for decision-making and compares for instance the two metrics "expected result" and "actual result".

The use of metrics supports measuring quality of software products and process. Metrics are applied to assess a product throughout the software lifecycle in order to verify whether the software quality requirements are met (IEEE 1061 1998). Moreover, metrics can be applied to assess the process quality in the development process (Satpathy *et al.*, 2000). The applicability of metrics in a software quality assurance programme will provide "a more disciplined, engineering approach to quality assurance" (Perry, 1991, p.541). For this reason, a process and product measurement concept is a component of this research.

2.4 Overview of Quality Approaches

The research in software quality can be divided into the two main strands. One focuses on techniques and approaches to assure the quality of software products (product-oriented approach) and another deals with software processes definition, evaluation and improvement (process-oriented approach) (De Oliviera *et al.*, 2002, p.671).

The aim of product-oriented approaches is to assure the quality of a software product by closely examining their significant characteristics. The most well known have been developed by Boehm *et al.* (1978), Gilb (1977) and McCall *et al.* (1977). Advocates of the product-oriented approaches say that to "clearly define, measure, and improve quality, one must measure characteristics of the software product which best influences quality" (Wong, 2006, p.66). Wong (2006) argues that product quality models view quality as inherent in the prod-

uct itself and characterise the model by their decomposition of product quality into diverse quality factors. A metrics model is used at the lowest level, which assesses the software quality to determine whether the product satisfies the quality needs.

Process-oriented approaches focus on the definition, evaluation and improvement of the software quality process, assuming that a high quality development process results in a high quality product. The model emphasises process improvement and the development of standards, arguing that effective processes are the key factor in producing high software quality (Wong 2006). However, good processes do not automatically imply a high product quality (Gillies 1997; Paulk *et al.*, 1993). It can be argued that good processes are necessary to improve the product quality, while the absence of standards will not help at all (Fenton 1996). The introduction of a process-oriented model is regarded as the foundation to achieve high product quality. This is supported by Paulk *et al*. (1994b, p.8) who argues, "the underlying premise of software process management is that the quality of a software product is largely determined by the quality of the process used to develop and maintain it". This means that software quality depends on the quality of the processes used to build it, which leads to the conclusion, the better the process the better the product.

Quality improvement approaches, such as the Plan-Do-Check-Act cycle of Deming (1988) focus on a continuous improvement of the product quality or the whole development cycle. Other improvement methods are Total Quality Management (TQM) or the Quality Improvement Paradigm (QIP), which claim that process improvements are only achievable if organisations learn from previous experiences.

Figure 2 depicts the different quality approaches assigned to the stated quality views.

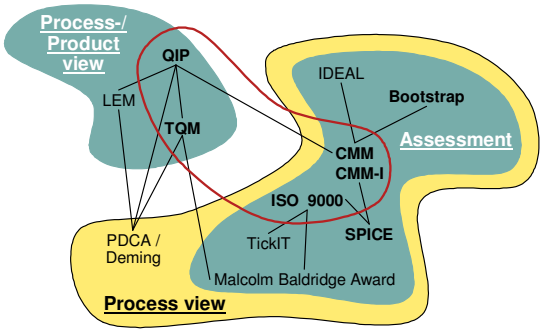

Figure 2. Overview of quality models (Rombach 2003)

A detailed overview of product-oriented, process-oriented and evolutionary quality approaches including examination of their associated models is presented in the following section.

2.4.1 Product Orientated Models

Product-oriented quality models commonly refine the abstract term “software quality” into more meaningful quality characteristics.

2.4.1.1 Introduction of hierarchical quality models

The establishment of a quality model is necessary to compare quality in qualitatively and quantitatively different situations. In the literature, many models have been suggested for quality and most of them are hierarchical in nature. Two principal models by Boehm *et al*. (1978) and by McCall *et al*. (1977) are still cited today. A hierarchical model of software quality based upon a set of quality criteria, each of which has a set of measures or metrics associated with it (Gilles 1997). Perry (1991, p.544) defines a schematically hierarchical view of software quality, as illustrated in figure 3. This describes a procedure to establish quality requirements as a three-level approach, corresponding to the hierarchical levels of a software quality model. The establishment of the software requirements or the goals on the highest level is the first step that needs to be performed.

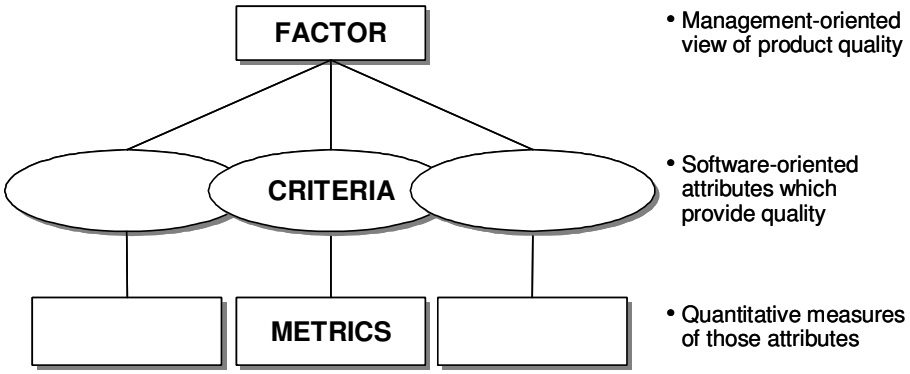

Figure 3. Framework for measuring quality (Perry 1991)

The highest level represents the management-oriented view of the product quality. The next level defines for each factor a set of criteria, which are the attributes that provide the characteristics represented by the quality factors. The lowest level consists of metrics that define quantitative measures. Quality requirements are unique for each project and need to be defined in relation to system or application-dependent characteristics. Perry provides several examples; if the system life cycle is long, maintainability becomes a cost-critical consideration, while if the system is changing frequently, the reusability of the main modules becomes an important criteria.

2.4.1.2 Quality Models by McCall and Boehm

The quality model by McCall *et al*. (1977) originates from the US military and is promoted by the Department of Defense (DoD). The model focuses on usage within the system development process and tends to define criteria reflecting users' and developers' priorities. McCall *et al.* (1977) address three major areas; product revision, product operation and product transition. The model considers three quality characteristics in a hierarchy of factors, criteria and metrics. The idea behind McCall's quality model is that the quality factors synthesized should provide a complete software quality picture (Kitchenham and Pfleeger 1996).

The quality model of Boehm provides a set of well-differentiated characteristics of software quality (Gilles 1997). Boehm's model attempts to define qualitative criteria by a given set of attributes and metrics. The model is similar to McCall but uses an extended hierarchy and subdivides further quality criteria. The high-level characteristic represents the basic software requirements of actual use of the system, such as as-is utility, maintainability and portability. The model considers two levels of actual quality criteria, the intermediate level being split into lower characteristics, which can be measured (Berander *et al.*, 2005).

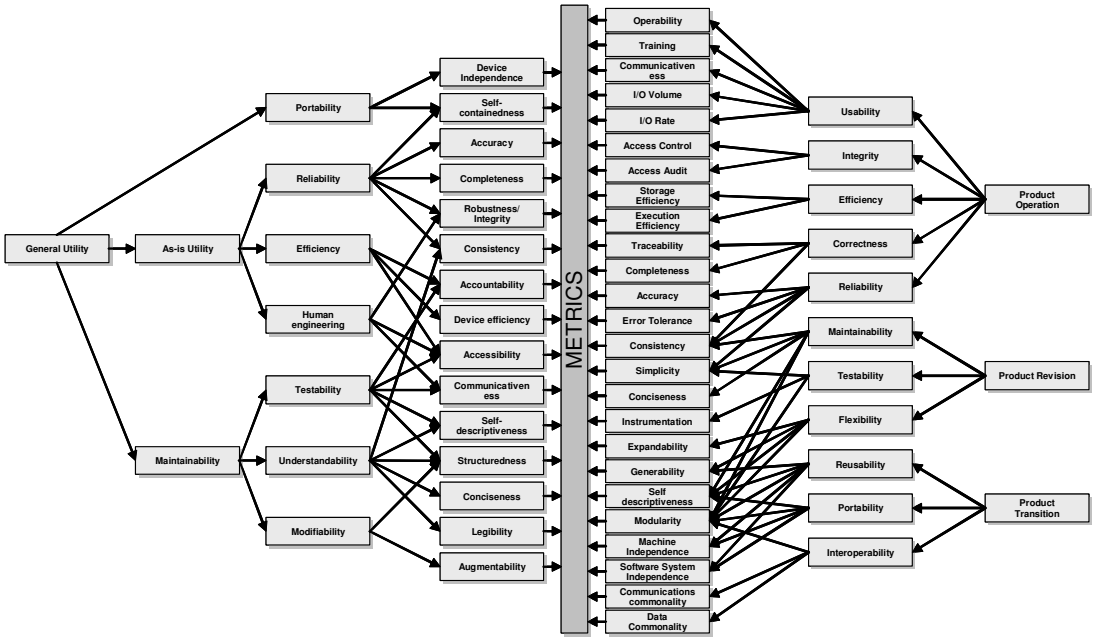

Figure 4. Quality models by Boehm versus McCall (Schulmeyer and McManus 1999)

The approaches of Boehm *et al.* and McCall *et al.* are similar although they differ in some constructs and metrics (Schulmeyer and McManus, 1999, p.414). Figure 4 depicts a comparison of both models, showing the quality model of Boehm *et al.* on the left and McCall *et al.* on the right side. Both models have been set up from an intuitive clustering of software characteristics and have a large degree of similarities or closely related characteristics. The

model of Boehm is based upon a wider range of criteria than McCall's model, but retains the same emphasis on technical criteria (Gilles, 1997, p.22). McCall's model focuses on the precise measurement of high-level characteristics. Both models share a number of common characteristics that Gilles (1997) compares as follows:

- Quality criteria are supposedly based upon the user's view
- Both models focus on the parts that designers can more readily analyse
- Hierarchical models cannot be tested or validated – it cannot be shown that the metrics accurately reflect the criteria
- The measurement of overall quality is achieved by a weighted summation of the characteristics.

2.4.1.3 ISO 9126 Quality Standard

In 1991, the International Organisation for Standardization (ISO) and the International Electrotechnical Commission (IEC) introduced the ISO/IEC 9126 standard that describes quality characteristics and guidelines for a software product evaluation. The standard aimed to define a quality model for software and a set of guidelines for measuring the characteristics associated with it (Cote *et al.*, 2004). Pfleeger (2001) reported some problems with the first release, such as how to provide an overall assessment of quality and how to perform the measurements of the quality characteristics; and rather than focusing on the user view of the software model's characteristics to reflect a developer view of software. The ISO committee began to rework the standards and the revised version of ISO/IEC 9126 was issued in 2004. The new standard ISO/IEC 14598 (ISO/IEC 1999) was published containing both the ISO quality model and inventories of measures for this model. ISO/IEC 14598 addresses Pfleeger's first concern, while the revision to ISO/IEC 9126 aims to resolve the second and third issues. The current version of the ISO/IEC 9126 consists of four standards:

- Quality model ISO/IEC 9126-1
- External metrics ISO/IEC 9126-2
- Internal metrics ISO/IEC 9126-3
- Quality-in use metrics ISO/IEC 9126-4

The software quality attributes may be grouped into external and internal quality attributes from the perspective of fitness-for-purpose and fitness-of-form (Fenton 1991). The ISO/IEC 9126 groups the different quality views into internal and external quality view and appends the quality-in-use view (ISO/IEC 9126-4 2001). Both, the internal and external quality have similar aspects and rely on a three-layer model, consisting of characteristics, sub-characteristics and metrics (see figure 5). The quality-in-use aspect aims at defining the quality attributes that are important for the end user (as shown in figure 6).

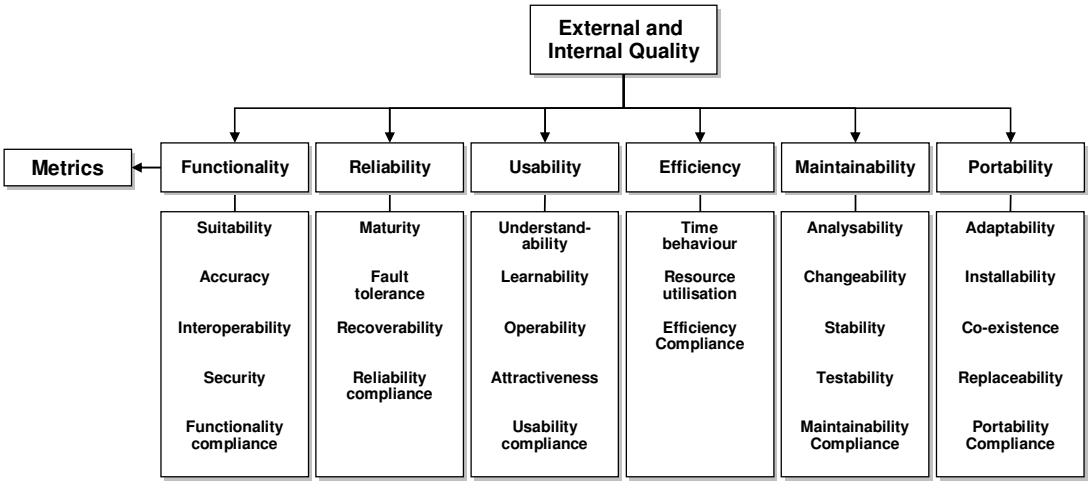

Figure 5. Three-layer model for internal and external software quality (ISO 9126-1)

The structure of the ISO 9126-1 was based on Boehm's and McCall's models. The ISO 9126-1 model includes functionality as a parameter and identifies internal and external quality characteristics of a software product.

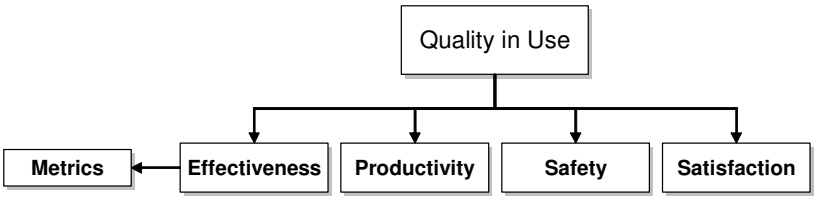

Figure 6. Two-layer model for Quality in use – (ISO 9126-4)

The different quality views are defined by ISO/IEC 9126 as follows:

- **Internal quality**: the totality of characteristics of the software product from an internal view. Internal quality is measured and evaluated against the internal quality requirements.
- **External Quality**: the totality of characteristics of the software product from an external view. It is the quality when the software is executed, which is typically measured and evaluated while testing in a simulated environment with simulated data using external metrics.
- **Quality-In-Use**: the user's view of the quality of the software product when it is used in a specific environment and a specific Context-Of-Use. It measures the extent to which users can achieve their goals in a particular environment, rather than measuring the properties of the software itself.

2.4.2 Process Orientated Models

The process-oriented approaches focus on indirectly improving the quality by assessing or improving the software development process. In recent years, the number of frameworks against which their processes are evaluated has multiplied (Sheard 1997).

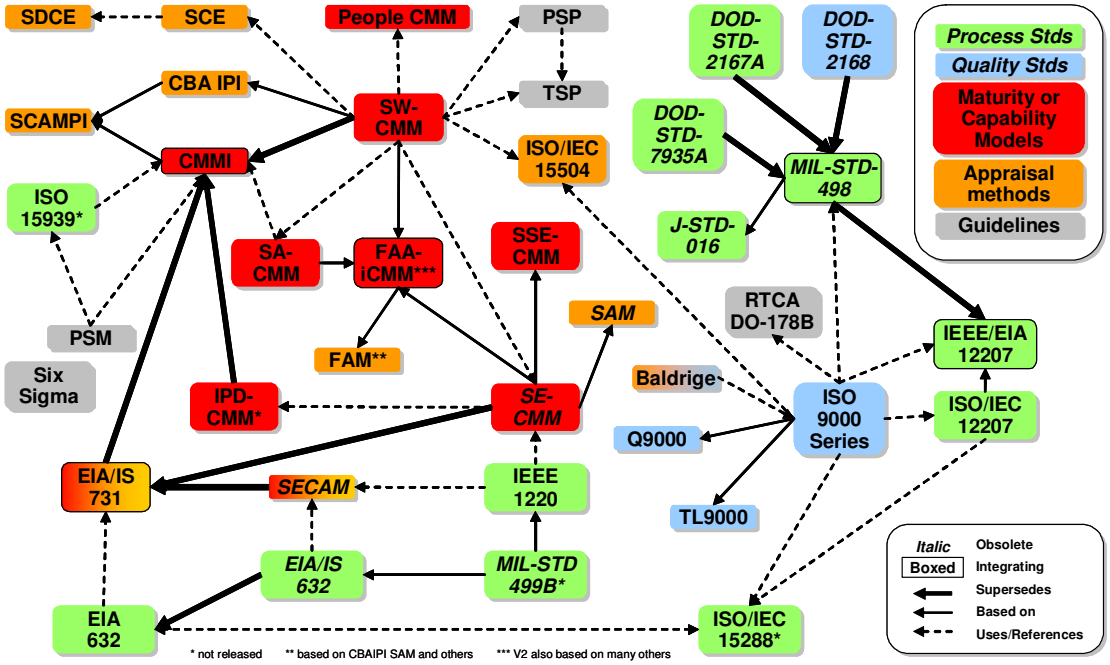

Figure 7. The Frameworks Quagmire (Sheard 2001)

As shown in the figure 7, Sheard (2001) classifies the frameworks by purpose and distinguishes standards and guidelines, process improvement models, Quality Awards, Software Engineering Lifecycle Models, such as ISO/IEC 12207 and Systems Engineering Models, such as SE-CMM, SECAM or the ISO 15288.

In the following sections, a selection of the important process-oriented approaches, such as ISO 9000 series, the Capability Maturity Model (CMM), the "Software Process Improvement and Capability Determination" (SPICE) and the Bootstrap model are examined.

2.4.2.1 ISO 9000 Series

ISO 9000 is a series of QA and management standards - guidelines for selection and use, developed by the International Organisation for Standardization (ISO) that defines a QA system for manufacturing and service industries. The ISO 9000 series comprise guidelines and standards for internal quality management purposes (ISO 9004) and for external quality assurance purposes (ISO 9001, 9002, and 9003). Paulk (1994, p.9) summarises the quality concepts as:

- Organisations should achieve a product or service quality, which meet the purchaser's needs.
- Organisations should "provide confidence to its own management that the intended quality is being achieved and sustained".
- Organisations should provide confidence to the purchaser that the intended product or service quality is delivered.

ISO 9000 standard specifies the quality system requirements for use between two parties for external or internal quality assurance purposes. ISO 9000:2005 describes the "Quality man-

agement systems - fundamentals and vocabulary" and covers the basics of quality management systems. Within this series, ISO 9001:2000 is a model for QA in design and development to assure the suppliers conformance to the specified requirements during the stages of software development and maintenance. Moreover, it includes a requirement for the continual improvement of the Quality Management System. ISO 9001 is a process-oriented approach. It internally addresses the organisation's processes and externally its management of the quality of delivered products and services. ISO 9000-3 provides guidelines for ISO 9001 for the development, supply, and maintenance of software (Paulk 1994).

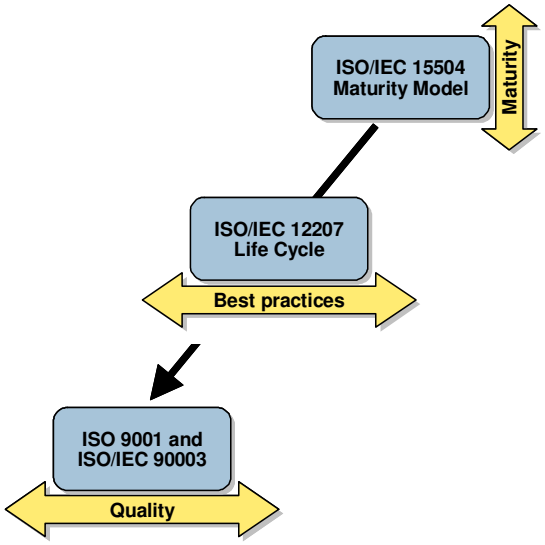

Figure 8. Relationship between ISO 12207, ISO 90003 and ISO 15504

ISO 15504 describes the software process assessment method, which is also known as SPICE, and refers to ISO 15288 (Systems Engineering - System Life Cycle Processes) and ISO 12207 (Software Life Cycle Processes).

2.4.2.2 ISO 12207

The international ISO/IEC 12207 standard for software lifecycle processes provides management and engineering processes for the software development. The standard is voluntary and is "designed for use by one or more parties as the basis of an agreement or in a self-imposed way" (Singh, 1998, p.1). The process model groups activities in the software lifecycle by "system context" and "software specific" processes and distinguishes seven process groups, as depicted in the figure 9.

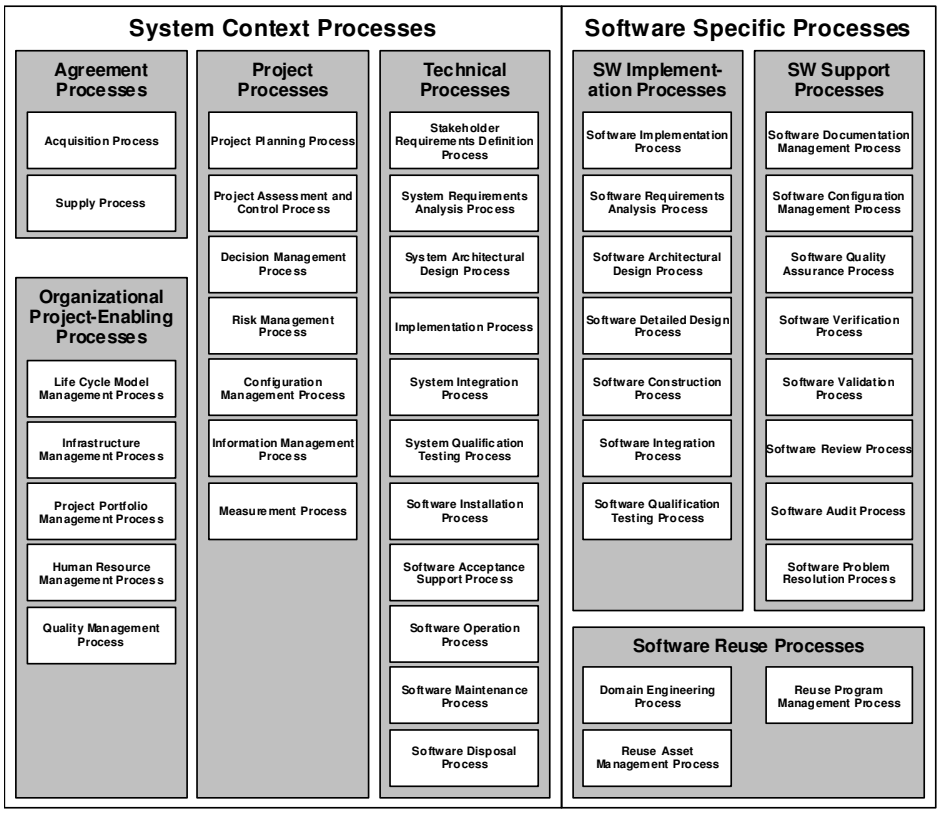

Figure 9. ISO/IEC 12007 Lifecycle Process Groups (ISO/IEC 12207:2007)

The model is structured on the highest level into processes, which cluster subsequent activities each of which consists of cohesive tasks. The ISO standards apply total quality management principles and treat all activities as an integral part of the lifecycle. Processes are equipped with a "plan-do-check-act" cycle, which assigns each process appropriate internal quality related activities.

The ISO/IEC 12207 standard has been developed for large and complex software projects. However, the approach is adaptable for smaller projects, which enables a broad usability. Singh (1998) states that the standard is applicable for any life cycle model, software engineering method or programming language. However, the ISO/IEC 12207 standard does not describe any metrics. It requires the definition of management indicators and software attributes and a reference to the ISO 9126 for guidance and quality characteristics (Singh 1998).

2.4.2.3 CMM

The CMM is designed to guide software organisations in selecting process improvement strategies by determining current process maturity and identifying the issues most critical to software quality and process improvement (Paulk *et al.*, 1993). The CMM is a framework to evaluate the maturity of the software development process focussing on continuous improvement. The Software Engineering Institute (SEI) has adapted the principles of product

quality developed by Juran and Deming into a maturity framework. It establishes a project management and engineering foundation for quantitative control of the software process as a basis for continuous process improvement (Paulk *et al.*, 1993). The CMM provides a framework divided into five maturity levels as shown in table 1. The maturity levels define an ordinal scale for measuring the maturity of an organisation's software process to evaluate its software process capability (Paulk *et al.*, 1993). Paulk describes a maturity level as a well-defined evolutionary plateau toward achieving a mature software process. Each maturity level comprises a set of process goals that must be completely fulfilled in order to obtain this level. The achievement of each level establishes different components in the software process, resulting in an increase in the process capability of the organisation. (Paulk *et al.*, 1993, p.24) For instance, an ISO 9000 certificated organisation is roughly equivalent to CMM level 3 (Tingey 1997). Table 1 shows the different levels with their respective Key Process Areas (KPA).

Table 1. CMM Maturity levels with corresponding focus and key process areas (Schulmeyer and McManus 1999)

Level	Level Description	Focus	Key Process Areas (KPA)	Result
5	Optimising	Continuous Improvement	Process Change Management	Productivity & Quality
			Technology Change Management	
			Defect Prevention	
4	Managed	Product and Process Quality	Software Quality Management	
			Quantitative Process Management	
3	Defined	Engineering Process	Organization Process Focus	
			Organization Process Definition	
			Peer Reviews	
			Training Program	
			Intergroup Coordination	
			Software Product Engineering	
			Integrated Software Management	
2	Repeatable	Project Management	Software Project Management	
			Software Project Tracking & Oversight	
			Software Subcontract Management	
			Software Quality Assurance	
			Software Configuration Management	
			Requirements Management	
1	Initial	Heroes		Risk

The CMM model overcomes the fundamental problem of organisations in their inability to manage software processes. Schulmeyer and McManus (1999, p.331) conclude that to achieve continuous improvements the organisation must become focused on a dedicated effort towards building a process infrastructure of effective software engineering and management practices. This results in a mature software organisation, that manages organisational wide software development processes, routinely communicates its processes, ensures that its processes are fit for use, defines roles and responsibilities, measures software quality

and customer satisfaction and works towards continuous improvement for judging product quality and problem analysis.

2.4.2.4 CMM-I

The Capability Maturity Model Integration (CMMI) integrates three models into a single improvement framework, which can be used by organisations pursuing enterprise-wide processes (CMMI 2006):

- Capability Maturity Model for Software (SW-CMM) v2.0 draft C,
- Electronic Industries Alliance Interim Standard (EIA/IS) 731 and
- Integrated Product Development Capability Maturity Model (IPD-CMM) v0.98

The CMMI model integrates system and software disciplines into one improvement framework and is structured around the following areas: Process Management, Project Management, Engineering and Support (CMMI, 2006, p.51). The CMMI built a framework that accommodates multiple disciplines and is flexible enough to support two different representations, such as staged and continuous (CMMI 2002). A representation is a different type of approach in presenting the capability maturity model and reflects the organisation, use and presentation of components in a model (Chrissis *et al.*, 2003, p.11). The CMMI distinguished two evolutionary paths within an organisation that are described by levels. The first path enables the organisation to incrementally improve individual processes. The second path enables organisations to improve a set of related processes by incrementally addressing a successive set of process areas. These two improvement paths are associated with two types of levels that correspond to the two representations (Chrissis *et al.*, 2003, p.73):

- Continuous representation with improvement path by capability level
- Staged representation with improvement path by maturity level

Figure 10 illustrates the two types of representation, which contain the same essential content and use the same model components (Chrissis *et al.*, 2003, p.74). The continuous representation uses capability levels to measure process improvement, while the staged representation uses maturity levels. The main difference between maturity levels and capability levels is the representation they belong to and how they are applied. The continuous representation has more specific practices than the staged representation. The continuous representation consist of two types of specific practices, basic and advanced, whereas the staged representation has only one type of specific practice (CMMI, 2002, p.20).

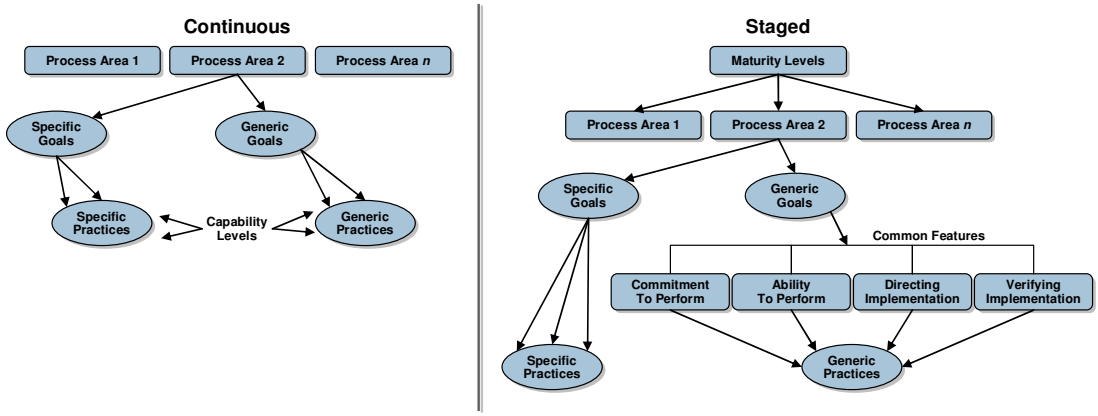

Figure 10. Continuous and Staged Representations (CMMI 2002)

Within the CMMI, capability levels belong to continuous representation and correspond with an organisation's process-improvement achievement for each process area. "Each capability level corresponds to a generic goal and a set of generic and specific practices" (CMMI, 2002, p.19). The CMMI considers six different levels, as shown in table 2.

Table 2. CMMI Capability Levels (CMMI 2002)

Capability Level	Continuous Representation Capability Levels
0	Incomplete
1	Performed
2	Managed
3	Defined
4	Quantitatively Managed
5	Optimizing

Maturity levels belong to staged representation and apply to an organisation's overall maturity (CMMI, 2002, p.19). Each maturity level comprises a predefined set of process areas, as shown in table 3.

Table 3. CMMI Maturity levels (CMMI 2002)

Maturity Level	Staged Representation Maturity Levels
1	Initial
2	Managed
3	Defined
4	Quantitatively Managed
5	Optimizing

2.4.2.5 Bootstrap

The aim of the Bootstrap project was to develop a method for software process assessment, quantitative measurement and improvement (Kugler *et al.*, 1994). The model follows the basic idea that software quality cannot be achieved effectively without defined methods and

organised processes within organisations. The method (as depicted in figure 11) is an improvement of the SW-CMM approach to process assessment and improvements, adapting the needs on software development organisations. Bootstrap assesses the development organisation and its processes, guidelines and implementation. The Bootstrap method has a wider scope of assessment activities, compared to the SW-CMM. While the SW-CMM model distinguishes only between yes and no answers, the Bootstrap questionnaire provides a fine tuned evaluation process, obtaining the score for the organisation. Instead of a unique score for the whole organisation, it allows an evaluation of a maturity level for each quality attribute. The Bootstrap maturity level algorithm determines quantitative process quality profiles for individual process quality attributes. This profile is the basis for decision making about process improvements and allows determination of the degree of satisfaction of ISO 9000 (Kugler *et al.*, 1994).

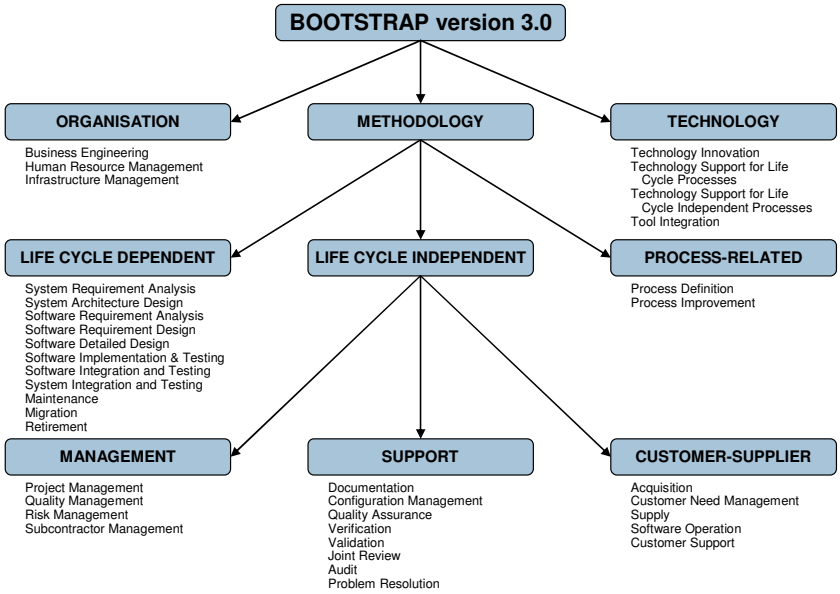

Figure 11. Bootstrap 3.0 process architecture (Bicego *et al.,* 1998)

2.4.2.6 SPICE

The SPICE project aims to establish an international standard for software process assessments for organisations of different sizes, application domains and management styles that may have different improvement priorities. The model is based on existing assessment methods, such as CMM, Bootstrap and ISO 9000-3. The results are defined in the ISO/IEC 15504 that consists of nine parts, covering concepts, process reference models and improvement guide, assessment model and guide, qualifications of assessors and guidance for determining supplier process capability. The ISO 15504 considers two main dimensions. The process dimension focuses on the improvement of processes and distinguishes several process categories. The capability dimension aims to determine the process capability ranked by capabil-

ity levels. ISO 15504 includes a process model with six capability levels that is applied to individual and reference processes. Process performance are evaluated using best practices. Process capability is built upon process assessment. Processes are rated against the process model, using a measurement and rating framework. The process assessment approach is depicted in figure 12.

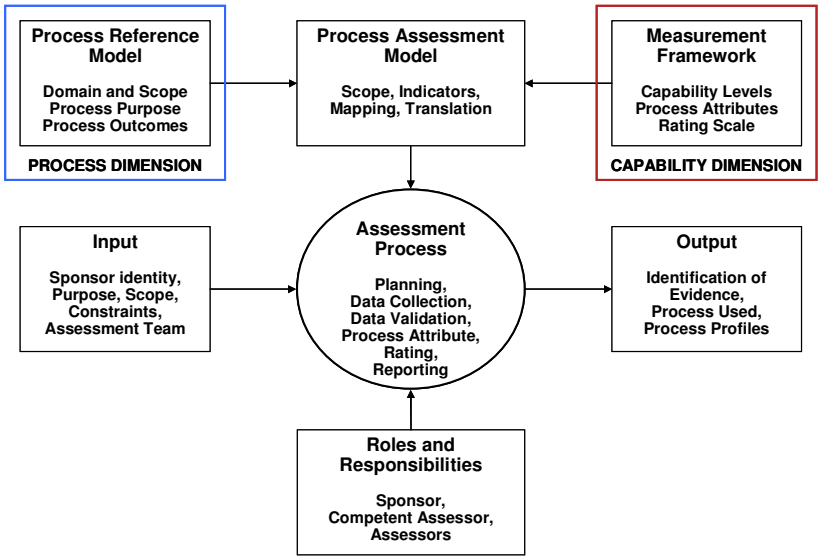

Figure 12. The Process Assessment Process (ISO 15504-5)

2.4.3 Quality Improvement Models

Quality improvement methods focus on the implementation of the improvement action. Most of the models are very simple and follow a stepwise or iterative approach. The Quality Improvement Paradigm (QIP) has a richer background, although the model focuses on the life cycle of a single improvement action (Kinnula 2001). Some classic improvement methods such as the Plan-Do-Check-Act (PDCA) cycle and the Quality Improvement Paradigm (QIP) are introduced.

2.4.3.1 Plan-Do-Check-Act (PDCA)

The Plan-Do-Check-Act (PDCA) was originally introduced by Shewhart in 1931 and is known as the Shewhart cycle. It represents a feedback loop for process improvements as depicted in figure 13. The status of the actual system is determined and the latest findings help in suggesting and executing improvement measures. Based on these improvements the system is rated in a new evaluation cycle. Each cycle starts with a planning phase to determine the aims and the measurements required to solve the problem. The suggested measurements are realised within the do-phase and the respective results are checked. Based on the result of the check-phase, the next iteration of the cycle is planned in the act-phase.

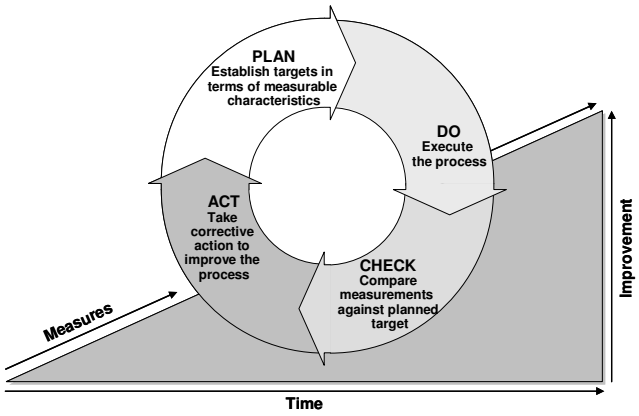

Figure 13. Continuous Improvement with Plan-Do-Check-Act cycle (Deming 1988)

The issue with this method is that continuous improvement focuses on consistent processes, tending to improve these constantly. Therefore, the method cannot be used for frequently changing processes. As software development projects are unique, this improvement method is only seen as a basis for the Quality Improvement Paradigm (QIP).

2.4.3.2 Quality Improvement Paradigm (QIP)

The QIP has been proposed by the Software Engineering Laboratory (SEL) to perform continuous process improvements (Basili 1985). The aim of the QIP model is to support continuous process improvement and engineering of the development processes (Basili *et al.*, 1994a). QIP combines engineering and empirical procedures and can be adapted to software engineering requirements. The QIP procedure considers each project as an experiment to obtain know-how and to re-use existing knowledge (Basili *et al.*, 1994a). The process consists of two feedback cycles, as depicted in figure 14. The project learning cycle delivers a feedback to the project within the execution step regarding performance information and reusable and improved software assets (Komi-Sirviö 2004). The six cyclical steps of the projects are as follows:

- Characterisation of project or environment
- Planning a set of goals, respectively and the methods to achieve them
- Selection of the appropriate process model
- Execution of the process, development of product
- Analysis of data
- Knowledge capture for further projects

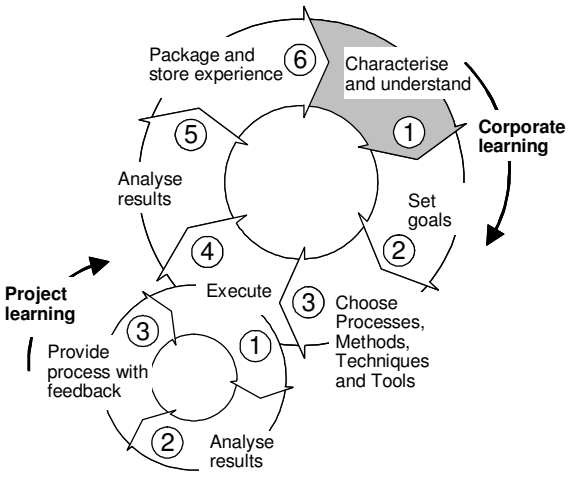

Figure 14. Quality Improvement Paradigm

The QIP shows how to set-up a continuous improvement process that uses previous experiences. The underlying assumption of this model is that the organisation is able to define processes, use tools and procedures for metrics and manage knowledge. Therefore, QIP can be used efficiently by mature organisations, with certain process knowledge, in order to become more sensitive to lessons learned.

2.4.3.3 Goal Question Metrics Paradigm

The Goal Question Metrics (GQM) paradigm (Basili *et al.*, 1994b, p.528) proposes a goal oriented measurement approach, which can be "applied to all life-cycle products, processes and resources". The top-down approach focuses on goals and models, which are evaluated by measurement. GQM follows the assumption that organisations and projects must define their goals, which are traced to the data that define these goals operationally. A framework supports the interpretation of the data in relation to the goals. The hierarchically structured GQM model consists of a conceptual, operational and quantitative level. The conceptual level defines the "goals", the operational level describes a set of "questions" to formulate the goals and the quantitative level defines a set of data to answer the questions in a quantitative way (Basili *et al.*, 1994b). The GQM approach of Basili *et al.* (1994b) is depicted in figure 15.

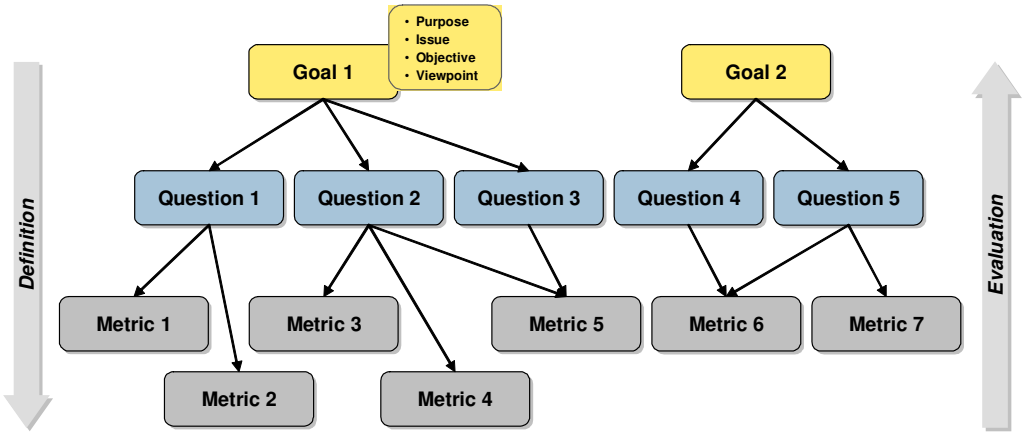

Figure 15. Goal Question Metric Paradigm

The goal specifies the purpose, issue, object and viewpoint of measurement. It is refined into questions, which are likewise refined into metrics. Some metrics can be used to answer different questions (Basili *et al.*, 1994b). The GQM model offers a generic approach to define and evaluate the measurement criteria for product or process quality measurement.

2.5 Software Quality Models: their potential effectiveness

Numerous definitions of quality can be found in the literature. According to Hoyer, two major emphases can be identified, one in which quality is seen as conformance to specifications and the other in which, quality is achieved when meeting the customer needs (Hoyer 2001). Garvin (1984) introduced five different views on quality, encompassing the best known quality models. Software quality assurance, assuring both the process and the product is required to ensure software quality (Fairley 1997). Software quality control is seen as the validation and verification processes, which focus on structured testing processes covering organisation, documentation and management (Reifer, 1985, p.20).

The execution of SQA activities requires the introduction of a quality model to obtain process transparency, customer satisfaction, repeatable processes and methods to ensure delivery of products in-time, in-cost and in-quality. The different quality models and procedures have shown that quality can be an elusive concept and can be approached from numerous perspectives. According to De Oliviera *et al.* (2002) quality models can be distinguished into product-oriented approaches that look for techniques and approaches to assure the quality of software products, and in process-oriented approaches that deal with software processes definition, evaluation and improvement. Either the models focus on process or capability levels such as ISO or CMM, or on a set of attributes and metrics to assess quality, such as Boehm or McCall. Those quality models have some advantages with regard to object measurability, as they are reducing the notion of quality to a few simple attributes. An essential part of a quality system is the Plan-Do-Check-Act cycle after Deming (Walton 1986). A quality improvement system which has a richer background is the QIP. Both models use feedback loops, where results can be used as input for further cycles to continuously improve the system.

Product oriented models, such as ISO 9126 offer a comprehensive range of quality characteristics and enable the evaluation of the product quality. However, focussing only on product quality characteristics will not guarantee mature products, if the underlying processes in an organisation are neglected (Grady and Caswell 1987 in Satpathy *et al.*, 2000, p.95). If processes are lacking in an organisation, the assessment of the product quality characteristics will reveal certain defects but the identification of the causes might be difficult. Process-oriented models offer processes and guidelines, which will help organisations to assess and mature their processes. For this reason, advocates of process-oriented models assume that by improving the process quality a mature organisation could produce mature products.

Several process-oriented approaches, which are completing the ISO 9001, such as CMM, CMMi, SPICE or Bootstrap were discussed in chapter 2.4.2. The ISO 9001:2000 model focuses on processes, considering product quality and measurements but does not provide further guidelines. The SPICE model, which is based on CMM or Bootstrap is defined in the ISO 15504:2004 and represents the assessment method for ISO 12207. The ISO 12207 standard provides a tailorable software development lifecycle model. Bootstrap assesses the maturity of an organisation (Kuvaja *et al.*, 1994). The CMM focuses on management, organisational and engineering aspects including product characteristics.

The quality models such as CMM, ISO 9001, ISO 12207, Bootstrap or SPICE enable a process assessment leading to improved process quality. Radice (2002) clearly shows the efficiency of a process model, identifying the relationship of a high CMM maturity level to defect detection effectiveness. Organisations, reaching the highest maturity level, have a defect removal effectiveness greater than ninety percent. However, Satpathy *et al.* (2000) state that these models are not specific enough to cover the requirement of individual processes. Satpathy *et al.* (2000, p.95) argue that the nature of processes varies widely, the model orientation focuses on the maturity of an organisation neglecting the overall process and often giving less importance to product quality. This research follows Satpathy's view that mature processes does not guarantee by in itself a quality product. Therefore, product quality characteristics need to be measured, as the implementation of mature processes does not implicitly reflect the product quality. Debou, 1999 (in Satpathy *et al.*, 2000, p.96) supports this argument and states that there is no evidence of a link between mature processes and high quality products.

Baker and Fisher (1999) introduced a Software Quality Program (SQP) that describes an overall approach to influence and determine the degree of quality achieved in a product. The SQP shows complex interactions between all levels involved and covers technical and management activities aiming to achieve a high product quality. Baker and Fisher (1999) conclude that the quality of a software product is dependent on the process used to build it. In consequence, poorly defined processes will not lead to repeatable quality products. Therefore, an evaluation of the software quality and associated documentation is required, based on a set of assessments and measurements for evaluation throughout the development process. Assessments are seen as qualitative evaluations, while measurements focus on quantitative evaluations. The evaluation of the product quality is difficult since the definition of software quality is hard to express quantitatively (Baker and Fisher, 1999, in Schulmeyer and McManus, 1999, p.122). The authors suggest establishing meaningful quality criteria based upon measurable entities that lend themselves to validation during the development process. The determination of which processes should be implemented needs to be carefully exam-

ined and cannot be based on the assumption "if a process has been shown to produce high quality software in the past then proper implementation of the process will result in a high quality product" (Baker and Fisher, 1999 in Schulmeyer and McManus, 1999, p.123). This argument is misleading as no conclusive link between the process and the resulting quality of the product itself can be found (Baker and Fisher, 1999). Therefore, quality assurance processes can only assure the proper implementation of the processes, which provide confirmation that the product meets established requirements. Dunn (1990) concludes, "SQA does not assure the quality of software, but the effectiveness of a software quality program (SQP)".

2.6 Software Quality Models: applicability for this research

Baker and Fisher (1999) assume that the quality of a product mainly depends on the process used to build it. Baker and Fisher, 1999 (in Schulmeyer and McManus, 1999, p.116) conclude, "software quality cannot be tested into a product. Quality can only be built in during the development process. Once the quality has been built in, the operating and maintenance processes must not degrade it". However, high process quality does not guarantee high product quality. Therefore, an overall framework suggesting processes and considering continuous quality measurement during the entire development lifecycle is suggested. It can be argued that a quality assurance framework must consider the processes and product view. A QA framework needs to focus on the assessment of product quality characteristics during the entire lifecycle, while the process model provides the underlying approach. The measurement of process quality shows the proper implementation of the framework, while the measurement of product quality criteria the conformance to specifications. This research follows the SQP approach and focuses on the establishment and control of procedures, as well as the product and process evaluation. A methodology, which describes processes in the software lifecycle has been discussed in the section on process oriented quality models. The ISO 12207 offers a standardised, tailorable life cycle model, which is flexible and adaptable to differing project sizes and needs. The ISO 12207 framework provides management and engineering processes for the software development and is selected due to its broad usability as a starting point for further investigations in the research area. Furthermore, the CMM is analysed as its model offers a process capability measurement approach. The evaluation of product quality characteristics as well as the process assessment requires the definition of a metrics model. As a basis for the evaluation of an appropriate product quality measurement the ISO 9126 standard is selected, which consolidates the views of e.g. Boehm *et al.* (1978) and McCall *et al.* (1977) to a worldwide standard.

3 Open Source Software: A Background

Open Source Software (OSS) has reached a remarkable popularity not least because of renowned products such as Linux, the Apache Web Server or the Mozilla web browser. These products have been developed under the OSSD model, promoting the free distribution and complete access to the source code (OSI 2005). In order to understand the origins of "Open Source", the underlying concept, the classification of software types, licensing models, the development approach, an introduction to the work is provided below.

3.1 History of Open Source Software

The origins of the open source model can be traced back to the UNIX development, which was developed at the AT&T Laboratories and was published in 1969 (Rosenberg 2000). Sharing source code between software developers, usually written in a higher programming language, was commonplace at that time. Due to a change of AT&T Licence policy for co-operative software in the early 1980s, UNIX became restricted to people who pay for the license. In 1984, Richard Stallman started a project to develop a free alternative of the UNIX operating system. Stallman established the GNU (named for Gnu's Not Unix) license, which was supposed to ensure that the software is indeed free and open for everyone (Rosenberg 2000). In order to support the GNU project the Free Software Foundation (FSF) was founded by Stallman in 1985.

A central element of this movement became the GNU General Public License (GPL), ensuring that software produced under this license remains free and promoting the production of even more free software. In the early 1990s, along with increasing use of the internet, many open source projects emerged. A renowned example is the unix-like operating system Linux, whose first version was released under the GPL licence by Linus Torvalds in 1991. The Open Source Initiative (OSI), founded 1997 by Stallmann and Perens aimed to promote OSS in commercial use, to allow business and Open Source Community to benefit from the OSS dissemination. The OSI developed the Open Source Definition (OSD), which is more like a trademark for OS software licences and guarantees several freedoms to software and commercial users.

However, recent success stories of many products developed under this model have given it enormous momentum and visibility, making it an interesting alternative for large software development companies (Behlendorf 1999). Pioneering the open source movement, Netscape was one of the first prominent companies who released their source code to the public in 1998. Companies, such as Apple, Corel or IBM tried different approaches to use, promote or

develop open source software. Since then, companies of all sizes have explored many economic models to succeed in the software market.

3.2 Understanding the Open Source Concept

The idea of the open source software is simple, although variants of the term exist, such as "free software" and "open source". The term "free software" originates from the GNU project and defines free software as relating to liberty, not price (FSF 2007). Stallmann introduced the term "open source" in 1998 to avoid a possible misunderstanding of the term "free software". Both terms describe almost the same category of software, but stand for views based on fundamentally different values as "Open source is a development methodology; free software is a social movement" (FSF 2007). In the following sections these terms are considered to be synonymous.

OSS is developed by individuals or companies who release their draft versions of the code to a community of developers. The community obtains an open-source license, which grants rights to use and to modify, compile and redistribute the source code "without a royalty or other fee" (OSI 2005). Open Source Software can be considered as a publicly available source code that can be used, reused and enhanced. In contrast to the classical software development that tries to prevent third parties accessing the source code, Open Source software is public and freely accessible. Unlike the "traditional closed software model", that allows only a few programmers to modify the source code, the open source model grants access to the source code to everyone for a rapid and evolutionary development process (OSI 2005). The freedom of the software does not imply that OSS must be gratis. Nor does the price charged for the software relate in any way to the price paid when the product was obtained (Working Group on Libre Software 2000). With traditional software, which is also defined as "proprietary software", the source code is not generally available and software is distributed often in binary form with a fee.

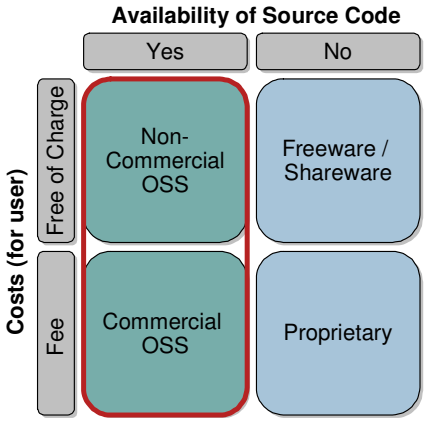

Figure 16. Classification of software (Wichmann and Spiller 2002)

Software can be classified according to license and distribution methods, as shown in figure 16. OSS can be distinguished according to the cost of the product into commercial and non-commercial OSS. Commercial software is often developed by businesses for economic reasons. However, a difference between commercial and proprietary software exists. For instance, most commercial software is proprietary, but there is commercial free software and there is non-commercial non-free software (FSF 2005). The non-commercial software is sometimes closed software that has been released for OSS. A rich landscape of software licences within the OSS area exists, differing in various aspects, such as right for commercial use or redistribution of the source code. Further examination of the OSS licensing system falls outside the scope of this research study.

3.3 Characteristics of OSS

The term "open source" is defined by the Open Source Initiative (2005) as a development model that allows free distribution and complete access to the source code. The distribution terms need to comply with the following criteria (OSI 2005):

- Free redistribution
- Availability of source code
- Allow modifications and derived work
- Keep the integrity of the author's source code
- No discrimination or restriction of the license

A main characteristic is the availability of the source code with the right to modify and redistribute the code free or with a charge. The transparency of the code offers the possibility of examining the code libraries and reuse of the desired parts. Developers can flexibly reuse and modify codes for their product, which becomes more efficient the higher the modularisation of development objects (Stamelos *et al.*, 2002; Aberdour 2007). The unrestricted access to the code grants insights into the program and allows inspections and verification at source code level. This is possible even in parts with closed software products, but the advantages

lie in the rights to modify the product according to one's needs (Working Group on Libre Software 2000). The rights to redistribute, use and modify the source code allow a large population to deploy the software without having to sign licenses (Peeling and Satchell 2001) and establishes a market for support and customisation (Working Group for Libre Software 2000).

The unrestricted availability of the code bears the risk that a developer could create an alternative code base and lead the development in another direction. This process is defined as "forking" and occurs for technical, license or management reasons (Working Group on Libre Software 2000). For instance, technical reasons exist, when one development direction focuses on the redesign, while the other stabilises the product. Another motivation for forking could be for management reasons, due to disagreements or poor management, or for licence reasons, where one release is published under a non-free license whereas the previous release is used to continue development under the OSSD model. Forking leads to direct competition of the products and has the effect that only a good quality product can maintain communities in the market (Working Group on Libre Software 2000).

The OSS model starts with an idea, followed by a more prototypical approach with frequent release cycles. The OSS momentum is driven by the participants' motivation, ongoing interactive tasks and continuous feedback from the community. A main characteristic is that the community defines the development direction in a democratically way. In contrast to traditional software, where vendors can decide to change the product, OSS users are not restricted in their usage and could even fund their own development to pursue their own direction (Working Group on Libre Software 2000).

The development involves a collaborative approach in a geographically distributed environment using web-based technologies. The geographic distribution challenges the communication activities to foster social interaction or to support social structures (Ankolekar *et al.*, 2003). In comparison to proprietary projects, where resources are planned and controlled, OSS projects bear the risk of unpredictability, as developers are participating on a voluntary basis. The OSSD model can attract bright and motivated developers who are not part of any corporate culture and offer an unlimited universe of programmers (Hoffmann 1999), and who can simultaneously review code and test an application (Raymond 2001). In contrast to traditional software, where the number of developers working together in a single team is limited, such limitations are not applicable to OSSD model. The OSSD model benefits from its community size and from parallel processes (Raymond 2001). Users are incorporated into the development process and can debug and test in parallel. Due to the availability of the code, users are not constrained to black box testing. The development method for software

harnesses the power of distributed peer review and transparency of process (OSI 2005). Peer-reviews are frequently applied in the OSS domain, which leads to constructive feedback and efficient defect handling (DiBona *et al.*, 1999, p.7)

Under the OSSD model a large community exists, which explains the huge number of beta testers (Peeling and Satchell 2001). The participation of large user communities in testing enables a wider variety of test scenarios than a single developer could generate (DiBona *et al.*, 1999, p.7). The model follows the principle of "given enough eyes, all bugs are shallow" (Raymond 2001) and is often coined as "bug-driven" development.

Dietze (2005) summarised the main characteristics of the OSSD model as:

- Collaborative development,
- Globally distributed actors,
- Voluntariness of participation,
- High diversity of capabilities and qualifications of all actors,
- Interaction exclusively through web-based technologies,
- Individual development activities executed in parallel,
- Dynamic publication of new software releases,
- No central management authority,
- "Bug- driven" development.

3.4 Open Source Development Model

A software development lifecycle provides a comprehensive picture of the steps involved in the development process. It provides methods, guidelines and practices tailored to the developers needs. However, there are differing views in the literature regarding the objectives and direction of the OSSD model. The simple development cycle as depicted by Hoffmann (1999) neglects process tasks and interactions within the team but contributes towards an understanding of the licensing model. The model described by Rothfuss (2002) concentrates on the management and organisational processes, whose interactions are constituted in a matrix. Dietze (2005) presents a UML-based OSSD model strongly focused on the requirements definition. In the model, Dietze distinguishes management, environment and development processes and shows complex interactions of identified roles, such as user, contributor, developer, reviewer and committer. Furthermore, Dietze (2005) points out that within the OSSD lifecycle a clear separation of the initial process and the improvement processes has to be considered. Therefore, Dietze characterised the main phases as:

- Initialization
- Gradual Software Improvement
    - Establishment
    - Advancement
    - Discontinuation

Figure 17 is a simplified presentation of the process intensity within the OSSD lifecycle according to Dietze (2005). It shows major processes only. The ideal gradient represents the intensity of the process but in reality is subject to variation.

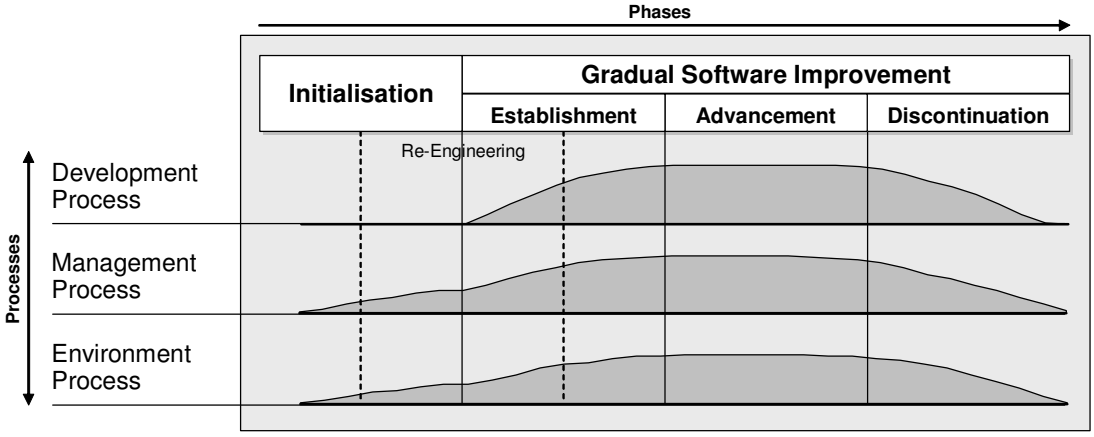

Figure 17. OSSD Lifecycle (Dietze 2005)

The initial development generally occurs in a self-contained process and has often a proprietary character. In this stage, the initiator develops the prototype, defines the appropriate licensing model and sets the foundation of the management processes required for the gradual improvement phase. The initiator is often part of the whole OSSD lifecycle, who participates in initial prototyping, development, maintenance and is represented on the management board. With the release of the product to the public, the initial phase is completed and the improvement process starts. It can be observed that after completion of the initial phase or the first iteration within the improvement phase a "re-engineering" is considered e.g. to adapt the system architecture to its new requirements or to the needs of a decentralised development process in a heterogeneous environment (Dietze 2005). The gradual software improvement phase is the main element of OSSD and contains the continuous distributed, parallel, stepwise refinement process. In the "Establishment" phase, the growth of the project is accompanied by an increase of management and infrastructure processes. These stagnate at a certain level in the "Advancement" phase, where the continuous improvement of the product occurs. The "Discontinuation" phase characterises the decrease of project activities until project end. However, a precise project end is indefinable (Ahmad and Lodhi 2006). The process intensity decreases with the attractiveness or the maturity of the product. Ahmad and Lodhi (2006) discuss the absence of a general accepted model and conclude that the challenges in defining such a model result from difficulties in capturing the development practices of dispersed teams, or different practices or varying organisational and cultural philosophies within the communities. They develop an OSSD model, which is similar to the QIP-life cycle as shown in figure 18. The model depicts the OSSD dynamics simply and provides a comprehensive picture of the process.

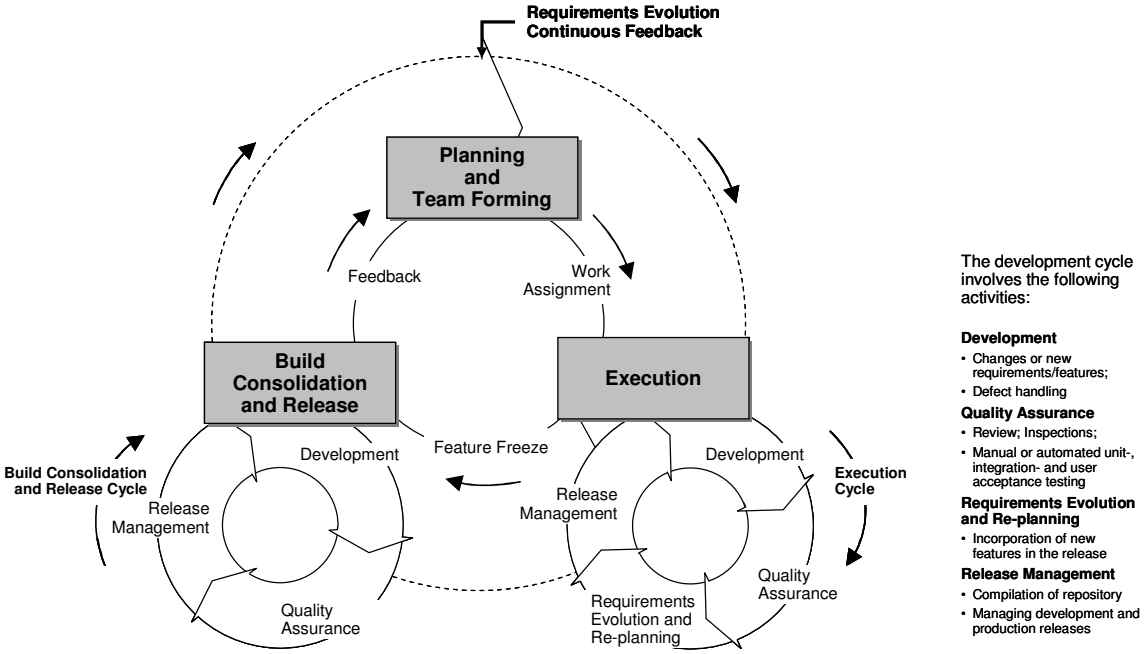

Figure 18. Open Source Development Model (Ahmad and Lodhi 2006)

The development in the OS domain occurs as an ongoing interactive task with frequent release cycles and constant feedback by the community. In the parallel development process, activities such as coding, editing, reviewing, testing as well as suggestion of enhancements or release management occur simultaneously. This is possible due to the large size of the community (Aberdour 2007). While developers invent new features, other users still test the product, report defects or new requirements and another member performs code reviews in order to integrate new parts into the product.

The planning activities in the initial phase, such as definition of requirements and team formation vary between the communities. According to Ata *et al*. (2002), those activities are much lighter than traditional planning techniques and involve the identification of a feature set for the current release, the modules that will be modified or built and the identification and assignment of responsibilities to the community members. In contrast to traditional software development, tasks are not directly assigned to developers, but individually selected according to interests or capabilities. In the execution phase, code development, reviews, defect tracking, planning of enhancements and release management activities are performed simultaneously. As the execution phase is mainly driven by the feedback of the users, the term “bug-driven” development was coined. The iteration stops when the release manager defines the scope of the new release and declares a freeze date. The freeze is necessary to stabilise the current release. This phase is declared as build, consolidation and release phase. No new features are incorporated into the release. The focus is to stabilise the software, e.g.

to complete the open developments, to perform testing and bug-fixing, and finally to announce the new release to the public.

3.4.1 Roles and Organisational Structures

Roles and responsibilities are clearly defined within the OSSD. Ata *et al.* (2002) describe the Apache projects and characterise the major roles of the user, developer, committer, release manager and the project management committee:

- The user deploys the products and contributes to the project by providing feedback to the developer regarding defects or new requirements.
- A developer is a user who contributes to the project by code or documentation, but has no right to commit the changes to the project repository.
- Committers are developers with the right to commit code or documentation to the repository. Committers oversee the development efforts among developers, perform code reviews, integrate code into the repository and ensure the integrity of the product.
- The release manager (RM) schedules the releases of the project. Thus, the RM decides the scope of the release, the execution of testing phases and scheduling when a release will be made public.
- The project management group controls the project and consists either of developers or committers with write access to the code repository, with voting rights for community-related decisions and with rights to propose an active user is promoted to the role of committer.

Members usually are nominated to their roles by the community or higher ranked members. Nevertheless, the organisational structure of OSSD is less strict. In some OSS projects the distinctions between core developer and users are informal and fluid, as participants can play different roles at different times (Mockus *et al.*, 2002). Jensen and Scacchi (2005) analysed the role migration process and observe similar findings. Compared to traditional software development, where "rigid and static organisational hierarchies with highly controlled growth at each layer" exist, roles are more fluid within OSSD (Jensen and Scacchi 2005). The structures of OSSD projects are hierarchical or "onion" like as depicted in the "onion" diagram in figure 19 (Crowston and Howison 2005)

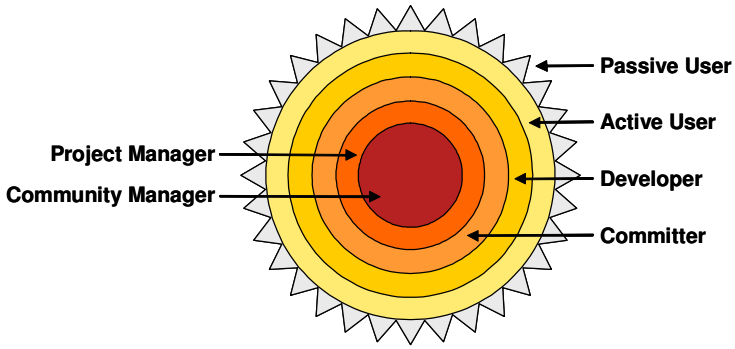

Figure 19. Organisational Hierarchy of an Open Source Community

The outer shell represents the passive users, who deploy the products but provide no contribution to the project. The virtual boundary to active users is the level of participation, in order to provide feedback about bug reports, testing or new releases. Rights and responsibilities are increasing towards the centre of the "onion". The inner shell represents the management of the community, which defines the general direction of the project. Crowston and Howison (2005) observe that community members tend to gravitate towards the central roles over time.

3.4.2 Cathedral versus Bazaar Style Development

Classic software engineering advocates argue that only a centralised management and strict access control to the source code lead to a quality product. Latest OSS development projects, such as Linux, Apache or Mozilla have shown that a large number of decentralised developers in a loose cooperation can deliver successful products with high quality. Similar to classic software development these projects incorporate a clear specification of requirements and rigorous management. In practice, several other projects can be found that generate successful products without a clear initial design or formal management approach (Working Group on Libre Software 2000).

Raymond discussed the unique aspects of the OSS model, such as the parallel activities within the debugging phase. Comparing the models for traditional software development with the OSS model, Raymond (2001) coins the terms "bazaar"- and "cathedral-style" development. The cathedral-principle is characterised by centralised planning and execution, top-down organisation with organisational hierarchies (analogous to how cathedrals were built in the Middle Ages). The bazaar-style describes a decentralised planning and execution, a network organised where the community of developers share the product and patches are exchanged via the Internet (like a bazaar where everyone can take what he wants). The cathedral-style development represents the traditional software development approach, which is also sometimes used for OSS projects. Open source projects can be approached with both styles; even blended development modes are possible. This occurs when projects are built in the cathedral-style until the first release and continue in the bazaar-style to allow the community to add more functionality. In the bazaar-style model, users are treated like co-developers. Due to the availability of the source code, users can easily diagnose problems, which leads to rapid code improvement and effective debugging. The Linux development has shown that "this effect would scale up with number and against system complexity" (Raymond 2001). Furthermore, Raymond observes in the bazaar style model a fluid and user-driven development of the source code and "loosely-coupled collaborations using the Internet." In the cathedral-model patches are strongly controlled, this includes the organisa-

tion of how they are integrated into the product, a defined management structure and a common release strategy. In contrast, the Bazaar-style model has a “loose” policy on code releases. Product releases are provided in high frequency but with an informal control as to who can provide what functions. The model follows the rule “Release early. Release often. And listen to your customers”. Long stable periods are available for general use, new functions and bug fixes are developed in parallel. In contrast to former development procedures, where high quality assurance procedures are applicable, the bazaar-model tends to release frequently and to distribute the code to the whole community. That allows developers and tester to check and improve the source code continuously. Raymond (2001) observes that the developers in the bazaar-model were constantly stimulated and rewarded by the prospect of “having an ego-satisfying piece of the action, rewarded by the sight of constant improvement in their work.” The maximised number of person-hours thrown at debugging and development gives enormous momentum in the open source development model. “Given enough eyeballs, all bugs are shallow” is the often summarised quote attributed to Torvalds (Raymond 2001). The debugging can take place in parallel and does not require intensive communication between the developers. Thus, it “does not fall prey to the same quadratic complexity and management costs that make adding developers problematic” states Raymond (2001). The accelerated debugging and code evaluation, which can be observed in the bazaar-style model, is explained by the fragmentation of developer and tester roles towards a mental model where both parties are looking into the source code. Thus, the source code awareness enhances both the communication and the synergy between developer and tester. As testers in the past could report only on surface symptoms, they now have access to the source code, which results in bug reports of a high quality (Raymond 2001). Therefore, the OSS development is often characterised as “bug-driven” approach. The bazaar model works like an accretion model, where products to gain a certain maturity level are adapted with more functionality and to get incrementally better. In this incremental development, a certain period exists where the product itself is unattractive, till it reaches the initial point of usability. Beyond this point, the product gains value from this accretion process.

The software development process under the OSSD model has many similarities with the proprietary or traditional software development. Nevertheless, both development types can be distinguished according to the characteristics as exemplified in table 4 (adapted from Wichmann and Spiller 2002).

Table 4. Differences of traditional software development to the OSSD model

Characteristics	Open Source Software	Traditional Software
Availability of Source Code	**Freely available**	**Not available**
Flexibility	**Adjustments of source code**	Adjustments by **customisation only**
Supplier	A **community**	A **company**
Stability	New features are incorporated if this **benefits the user**	New features are incorporated if there is a **commercial incentive**
Development Principle	"**Bazaar**"-principle" (often blended; Cathedral till first release afterwards Bazaar)	"**Cathedral**-principle"
Development Direction	Direction of **development decided by the user/team**	**Clear functional** requirements
Emphasis of Development	Tend to emphasise **stability** and **functionality; "user"-driven**	Emphasise **usability,** driven by **corporate economies**
Debugging	Debugging within the **whole community**	**Limited** recourses for development and debugging
User participation	User **participates in communities** and influences the development direction	user commonly **not organised;** maintains contact to supplier independently
Team Size	**Large group** of freely participating developers; level of user participation is extremely **high**	**Small** development team, developer paid for the supplier
Team Structure	**Hierarchically** structured team but less restrict	**Development cycle** for standard or individual Software

Traditional or proprietary software is typically delivered in binary form (as closed source), which restricts modification only to customisation settings. Flexible adjustments of the product are difficult and lead to high development cost and effort. The availability and the access to the source code is a precondition for a community wide OSSD process, including testing and debugging activities. A small number of developers usually perform debugging in proprietary development projects, while parallel debugging in OSS projects benefits from their community size. In contrast to OSS projects, where requirements definition is not restricted and executed independently to release or patch processes, the planning in traditional development projects is aligned to the customer requirements. This is defined in a traditional project, the team size is planned and can be extended if necessary. On the other hand, the OSS developer freely participates within the project, which risks uncertainty about project continuation and restricts any resource planning and scheduling. The OSSD team organisation is hierarchically structured similar to traditional projects, but handled less rigidly. Another distinction can be found in the development direction. According to Wichmann and Spiller (2002, p.26) proprietary software tends to place emphasis on usability where OSS emphasises stability and functionality because it is aimed at a different audience.

3.4.3 Comparison of Development Methods

The OSSD can follow different development methods, although the method must comply with the requirements of the OSS model (Abrahamsson *et al.*, 2002, p.80). It is important to understand that the OSSD can use agile as well as plan-driven aspects. According to Boehm (2002) "plan- driven" approaches are another description of process-oriented methods. Agile methods are characterised by Boehm (2002, p.64) as often less plan orientated than they really are, because their general emphasis is more on planning than on the resulting documentation. Agile methods are an attempt for a faster and nimbler software development process.

Boehm (2002) analyses and compares agile and plan-driven approaches, augmented by Abrahamsson (2002) with the OSSD model as shown in table 5.

Table 5. Comparison of OSSD with agile and plan-driven methods

Home-ground area	Agile methods	Open Source Software	Plan-driven methods
Developers	Agile, knowledgeable, collocated, and collaborative	Geographically distributed, collaborative, knowledgeable and agile teams	Plan-oriented; adequate skills; access to external knowledge
Customers	Dedicated, knowledgeable, collocated, collaborative, representative, and empowered	Dedicated, knowledgeable, collaborative, and empowered	Access to knowledgeable, collaborative, representative, and empowered customers
Requirements	Largely emergent; rapid change	Largely emergent; rapid change, commonly owned, continually evolving - "never" finalized	Knowable early; largely stable
Architecture	Designed for current requirements	Open, designed for current requirements	Designed for current and foreseeable requirements
Refactoring	Inexpensive	Inexpensive	Expensive
Size	Smaller teams and products	Larger dispersed teams and smaller products	Larger teams and products
Primary objective	Rapid value	Challenging problem	High assurance

The OSSD has many similarities with agile methods. A classification of the OSS methods in contrast to agile and plan-driven methods is shown. Although "OSS is not a compilation of well defined processes and published software development practices constituting a written eloquent method" (Abrahamsson *et al.*, 2002, p.74), it follows the practices of other agile methods, such as the frequent release cycles or the collaborative development approach. However, Cockburn (2002) says that the OSSD differs from agile methods in philosophical, economical and team structural aspects.

Although OSSD has many similarities to agile methods, this research considers OSSD as a development methodology, with more tendencies to process-oriented approaches. Large projects face increased complexity and communication costs, which rise exponentially with the number of developers (Brooks 1995). Thus, another weighting of processes and methods to support these projects is considered, which is of interest in the following research.

Agile methods are seen by Constantine (2001, p.69) as an attractive approach for small projects. He concludes that the "tightly coordinated teamwork needed for these methods to succeed becomes increasingly difficult beyond 15 or 20 developers". Cockburn and Highsmith (2001, p.133) report that the average size for agile projects consists of nine people and conclude that agile development is more difficult with larger teams, due to increased collaboration effort. Plan-driven methods fare better for large projects. However these approaches are often more bureaucratic and require additional organisational effort, which will not be very efficient on small projects (Boehm, 2002, p.67)

3.5 Quality Assurance under the Open Source Software Development Model

The OSSD has increased significantly in recent years, for reasons beyond the low costs in the easy code access and the consistently high software quality (Raymond 2001). Many applications developed under the OSSD show levels of quality comparable to or exceeding that of software developed traditionally (Halloran and Scherlis 2002). Raymond (2001) argues that the high level of quality is partly due to the high degree of peer reviews and user involvement observed in OSS projects. The software development method harnesses the power of distributed peer review and the transparency of the process (OSI 2005). However, the empirical evidence that the OSSD model results in high quality, high reliability and flexibility is not available (OSI 2005). Empirical studies about quality assurance activities are rare (Glass 2001), which makes it difficult to support or deny Raymond's position. In recent years the research about QA activities under the OSSD model has increased with studies by Zhao and Elbaum (2000, 2003), Halloran and Scherlis (2002), Koru and Tian (2004), Michlmayr (2005) or Aberdour (2007). The recent findings and observations of QA aspects are summarised under the following headings:

- General Criteria
- Human Resource Aspects
- Documentation
- Communication
- Development Processes and Methods
- Testing and Defect Handling
- Infrastructure Characteristics

3.5.1 General Criteria

Important success factors under the OSSD model are the project community size and the level of user participation (Aberdour 2007). High user participation supports the permanent feedback loop from early project stages onwards to continuous review to meet user requirements. Developers benefit from high user participation, due to expanded capacities and shifting of tasks (Lerner and Triole 2002). Zhao and Elbaum (2003) confirm the high user participation and find in more than half of the surveyed projects, user groups of more than fifty people.

Complexity

Brook's (1995) law predicts that the complexity and the communication costs of a project rise with the square of the number of developers, while work done only rises linearly. Therefore, larger OSS projects face high challenges, as there is an exponential growth of project communication and co-operation with the increase in project size. The complexity of these processes also increases with the project size, while debugging and testing processes can be undertaken in parallel and benefit from larger communities (Raymond 2001).

Defect location

An analysis of the defect location on OSS projects indicates that interface and logical errors are the most common defect types (Zhao and Elbaum 2000). Furthermore, a relationship between defect location and programming language could be observed, where C++ applications have a greater percentage of logical defects and Java applications have more problems with network issues than other development languages (Zhao and Elbaum 2000).

Development language

Zhao and Elbaum (2000) hypothesise that project quality may be influenced by the development language. Their study shows the longer languages exist, the more testing tools are available that could significantly improve the development process. Thus, the applicability of mature development languages indirectly contributes to quality.

3.5.2 Human Resource Aspects

Developers are freely participating and intrinsically motivated in OSS projects. However, if there is not enough interest, there is no guarantee that development will happen (Peeling and Satchell 2001).

Motivation

Hars and Ou (2001) observe a high external motivation in OSS projects and find that personal needs are another key factor not yet receiving sufficient attention. Zhao and Elbaum (2003) examine developer motivations and observe that a high personal motivation exists. However, this tendency becomes less obvious with the growth and maturity of the projects (Zhao and Elbaum 2003). The reward-and-recognition culture contributes to personal motivation and facilitates the creation of a sustainable community (Aberdour (2007).

Developer experiences

OSS projects seem to benefit from experienced developers. For instance, Zhao and Elbaum (2003) find that the majority of developers have more than five years software development experience. Massey (2003) argues that successful medium and large OSS projects tend to be designed by developers of extraordinary skill and experience.

Such level of experience might influence the product quality significantly (Putnam and Myers 1991; Reifer 2004). Knowledgeable developers may approach things correctly, based on their experience and follow established practices. Thus, less guidance and coordination may be required to manage such collaborative development activities. Following this assumption, a key element for high product quality is the presence of experienced developers.

Voluntary participation

However, OSS projects face a challenge to achieve a constantly high level of quality when the voluntary participants continually change in an unpredictable fashion (Michlmayr 2005a). Resources changes, such as joining or leaving developers are usual in software projects, but the fact that OSS developers participate as volunteers attaches greater importance to this topic than in traditional software development projects. Traditional development projects are characterised by more stable resource planning. The unpredictable participation in OSS projects bears the risk of a lack of knowledge and time-consuming effort to attract new volunteers (Michlmayr *et al.*, 2005). Hence, OSS projects need to consider closely knowledge capturing and aspects of integration.

3.5.3 Documentation

User and developer documentation are important quality factors in OSS projects, but are often neglected (Michlmayr *et al.*, 2005). The user documentation influences the motivation, as it supports information retrieval and knowledge transfer. The developer documentation defines, for instance, coding style and commit activities, which have direct influence on the quality of the development process. The process of taking new developers on to projects is

supported by appropriate documentation relating to the product, the development approach, the methods and community organisation. Shortcomings in documentation significantly reduce the contribution level of each developer. For instance, a lack of documentation may lead to an increased effort for information retrieval, which negatively influences the developer motivation.

Peeling and Satchell (2001) observe that OSS products are often well structured with good documentation but tend to get the ease-of-use features and the user-oriented documentation later than commercial products. Zhao and Elbaum (2003) find that documentation do not play an important role in OSS projects, as mainly "to-do"-lists or installation guidelines are used, but design documents or release plans are lacking. Michlmayr *et al.* (2005) state many developers criticise a lack of documentation regarding development practices and that only a few projects have explicit documentation describing how to contribute to the project. Furthermore, a relationship between documentation and project size seems to exist. Those projects with large numbers of contributors generally have appropriate documentation that describe the internal processes, such as coding style or commit practices (Michlmayr *et al.*, 2005). General project information, design documents or tasks lists are essential (Aberdour 2007) to capture the project knowledge and to support the integration of new resources into the project. Documentation necessary for development projects is often underestimated but has significant impact on the product and process quality.

3.5.4 Communication

Collaboration in a geographically dispersed environment is difficult due to the lack of communication. "Fostering social interaction and supporting the social structures within the open source software community would encourage informal communication and provide a context for community members to interact and share information", argues Ankolekar *et al.* (2003). Small projects teams benefit from the efficiency of direct communication and low complexity in decision-making structures, while medium and large sized projects face problems with coordination and communication (Brooks 1995; Michlmayr *et al.*, 2005). Unclear responsibilities, missing contact persons or lack of communication related to development processes or defect handling could lead to ineffective or duplication of work. The effective communication between developer and maintainer is a prerequisite for the development process. Strong co-operation of maintainers with the upstream authors, who reuse software from other sources, is required to communicate defect reports and feature requests (Michlmayr and Senyard, 2006, p.145).

3.5.5 Development Processes and Methods

According to DiBona *et al.* (1999, p.17), the "OSSD model can lead to a faster development time, since OSS projects can develop and debug new software with the speed and creativity of science". However, working in parallel has its restrictions, while some tasks increase with size. Similar to Brooks (1995) renowned statement "adding developers to a late project makes it later", Michlmayr and Hill (2003) argue, "merely adding further programmers to a project will neither improve its quality nor shorten the release cycle". The development process in OSS does not benefit from the concurrent working in the same way that debugging or quality activities do. In fact, development tasks are not scalable as debugging or quality assurance activities, as these activities do not require as much inter-personal communication as software development does (Porter *et al.*, 2006).

Software development processes in OSS projects are often informal, which is seen as an advantage by some researchers. Peeling and Satchell (2001) argue that OSS developers are not constrained by corporate product development processes and QA processes. The authors see benefits in less restricted processes, as opposed to proprietary software development projects, because the achievement of a high product quality is the overall goal under the OSSD model. Moreover, Massey (2003) argues the OSSD model is unique and will not benefit from process maturity or traditional insights. Villa (2003, p.10) contradicts Peeling's and Satchell's position and concludes that QA activities are even more effective when the people are interested in the process and not just in the results. Most of the well known OSS products have a "rate of evolution, robustness and responsiveness to bug reports that much commercial software can only dream of" (Peeling and Satchell 2001). Thus, processes need to be defined and applied community wide to support the collaborative development activities (Michlmayr *et al.,* 2005). Aberdour (2007) concludes, "the system and the community must co-evolve to achieve sustainable development and high-quality software".

Design

In traditional software engineering, the design process is performed as a combination of top-down and bottom-up approaches. The OSSD follows an alternative approach, where "a small amount of middle-out design concentrated on a layer somewhere just below the top, followed by extensive design-by-coding" (Massey, 2003, p.93). Missing or unspecified design documents in OSSD process are a risk for the project success and the software quality (Aberdour 2007). Uncertainties in design documents influence the quality regarding product operation, transition or product revision criteria and may require a complete redesign of the system architecture when requirements evolve. Requirements may change in an unpredictable way under the OSSD model, as the development direction is user or community driven.

Even the scheduling of the development of new features is difficult to plan and brings uncertainty (Peeling and Satchell 2001). OSS projects face the problems that design documents often lack detail and effective communication regarding development priorities does not exist (Villa 2003). Aberdour (2007) concludes that well documented design and high code modularity contribute to high software quality.

Code Coverage

In the OSSD model, knowledge is spread throughout the community and users do not rely on a single commercial organisation (Peeling and Satchell 2001). The dependence on a single vendor is significantly reduced. For instance, if a single vendor stops the support of the product, another one can easily continue and there are many knowledgeable developers who can offer support. Peeling and Satchell (2001) state that commercial software may become un-maintainable once its originators leave the company and they conclude that the lifetime of OSS products is much longer since it is community property. However, certain quality problems exist in the development process due to unsupported development code (Michlmayr *et al.*, 2005). OSS projects face problems in handling source code that has previously been contributed and which is now unmaintained, because the original contributor has left the project. Any further development in this area, which requires an update of that specific code, leads to severe problems and extra work for the development team.

Release Strategy

The release strategy is an important criterion and has an impact on the software quality in one of two ways (Michlmayr 2007). A feature-based strategy focuses on the readiness and maturity of the features, while a time-based approach follows a schedule. Maturity-based release management reduces conflicting priorities from marketing processes, as the product is delivered when “it’s ready”, which results in lower maintenance costs and less updates or patches (Working Group on Libre Software 2000).

The release frequency is a sensible control element that affects quality and user satisfaction. Michlmayr (2005) observes that a clear release strategy coupled with testing contributes to a successful project. Frequent release cycles allow quick consideration of bug fixes or integration of user requirements. This influences the user feedback significantly. Rapid release cycles keep code reviewers and developers interested and motivated, resulting quickly in new features and high quality (Aberdour 2007).

The frequency of the release cycles need to be scaled to a level of intensity that matches the complexity of the development. Michlmayr and Senyard (2006) conclude that frequent release cycles with close interaction of upstream authors lead to shorter defect removal times

and more user feedback. This contradicts to Porter *et al.* (2006) who argue that short release cycles leave less time to build stable versions, which could lead to users' frustration. Thus, the optimum release frequency needs to take account of stability while providing an adequate feedback. Zhao and Elbaum (2003) find that 43% of projects release a new version every month, which provides further evidence that the OSS development approach follows the often cited statement: "Release early, Release often" (Raymond 2001).

The release management activities of OSS projects face issues, regarding management skills, developer commitment and the release characteristic (Michlmayr 2005a). The responsible release manager needs management and co-ordinating skills. In order to achieve a milestone, extra effort is often required, which volunteers may not be able to contribute. Because participation is voluntary, projects cannot rely upon a consistent level of work or extra work to deliver a release (Michlmayr *et al.*, 2005). Developers do not see the necessity to build a stable release for less technically oriented users, since developers work with development releases anyway. Inadequate release management can lead to several problems, such as incompatible software, or software which does not meet the quality standards or the user requirements (Michlmayr 2005a). Insufficient release scheduling has the problem that consistent software quality cannot be ensured, due to little time or short feedback loops between users and core developers. This typically results in frequent "beta" releases (Porter *et al.,* 2006). Frequent releases offer quick responses to defects but frustrate less technically oriented users who require stable versions.

3.5.6 Testing

The average testing time during a software maintenance project averages 13% to 24% of the total development time (Basili *et al.*, 1999). Similar findings for OSS projects are reported by Zhang and Pham (2000) and Zhao and Elbaum (2003). Larger projects spend on average less time on testing than smaller projects (Zhao and Elbaum 2000, 2003). Half of the examined projects follow structured testing approaches using a baseline test suite and testing plans (Zhao and Elbaum 2003), which influences the defect solution rate. Structured testing approaches are quite demanding of resources and unrewarding for participants (Massey 2003). Under the OSSD model, testing activities are shifted to the community and projects wait for the users to report defects. This approach is supported by the fact that many of the users are professional developers who can understand the code and provide accurate failure reports (Massey 2003). Furthermore, this might explain the high bug reporting quality observed by Zhao and Elbaum (2003). High modularity and many professional bug finders and fixers result in low defect density (Aberdour 2007). Debugging benefits from concurrent testing activities, as larger projects seem to profit more from user testing with as high testing effi-

ciency as small projects (Zhao and Elbaum 2003). Aberdour (2007) conclude that projects can rely on the user base for system testing but need to give sufficient resources and sponsorship.

Test Efficiency

DiBona *et al.* (1999, p.17) argue, "due to the large community of users that participate, testing results are more reliable and enable a wider variety of test scenarios than a single developer could generate". OSS projects seem to follow the principle of "given enough eyes, all bugs are shallow" (Raymond 2001). However, complex testing processes face problems, when the product is platform independent or offers flexible customization. Porter *et al.* (2006) argue, since core developers may have access to a limited number of platform configurations, they release code that is not tested in all environments. Michlmayr *et al.* (2005) assert that it is nearly impossible for the lead developer to test all customising combinations, with the effect that users report that new releases break their configuration. Furthermore, it can be assumed, that users are only testing their desired settings, which means that an uncertain gap of untested scenarios may exist. A lack of test scenarios or the inability to execute broad regression testing due to high software complexity could affect the software quality tremendously. Aberdour (2007) suggests completing structured testing approaches with formal testing techniques and regression test automation.

User Suggestions

Aberdour (2007) concludes that high quality in OSS relies on having a large sustainable community, which results in rapid development, effective debugging and promotion of new features. However debugging plays a more important role than user suggestions (Zhao and Elbaum 2003). These observations vary depending on the project size. Small projects incorporate user suggestions more easily, which indicates greater flexibility in smaller projects.

Specification Reviews

OSS projects put more emphasis on field-testing and user-reviews than traditional software development and take advantage of the users' willingness to work with an imperfect product (Vixie 1999). The OSSD model offers high potential as its collaborations lead to high levels of peer review and user involvement (Raymond 1999). The applicability of peer-reviews leads to constructive feedback and efficient defect handling (DiBona *et al.*, 1999, p.7). Under perfect conditions, the OSSD model successfully employs different testing techniques and develops very high quality products (Tuma 2005).

Studies demonstrate that code reviews can remove defects more efficiently than testing (Tuma 2005). In closed software development, only internal developers can perform code

review and users are constrained to "black-box testing". In OSSD, this barrier is offset due to the incorporation of the user into the development process. Code reviews, which are performed by people outside the project team leads to independent, objective reviewing (Aberdour 2007). Zhao and Elbaum (2000) find that on average, reviews and inspections of the source code are done 1.3 times per application in all projects, while up to 7.5 reviews per application could be undertaken in large projects. This may show that the sheer mass of participants has a significant influence on inspections. It is interesting to note that around 75% of the respondents declared that they did not ask anybody for a review (Zhao and Elbaum 2000). Hence, code inspections and reviews seem to be a well-established process in OSS projects. Zhao and Elbaum (2000) find that there is significantly more time spent on source code inspections than on design or requirements inspections, but large projects tend to spend on average more time in inspections than smaller projects.

Quality Control before Commit

Code commit practices differ between projects, as some projects allow commit access with the first submission, while in other projects only the lead developer can submit code (Zhao and Elbaum 2000). Thus, the impact of different code control practices on software quality needs to be investigated.

3.5.7 Defect Handling

The defect handling processes under the OSSD model benefit from direct openness of the code and the ability of users to fix defects by themselves, which allows them to control software vulnerabilities (Peeling and Satchell 2001). The duration for defect handling is significantly reduced, as "bug fixes have been shown to come out about six times faster for OSS than for proprietary equivalents" (Nowak 2003). Despite the advantage of collaborative debugging, the defect handling processes faces certain quality challenges, such as the bug reporting quality.

Defect Reporting Quality

The quality of defect reporting suffers if no broad understanding of the architecture of the entire system is available. Users, who do not understand the system constraints may suggest inappropriate fixes, argues Porter *et al.* (2006). The severity of this problem increases, as users do not know how to report bugs. "As more users with few technical skills use free software, developers see an increase in useless bug reports", states Michlmayr *et al.* (2005). Ineffective bug reporting, due to duplicate records, insufficient description or ignorance of guidelines reduces the effectiveness and efficiency of defect handling.

Debugging processes could suffer from lack of analysis of duplicate entries and lack of information about priority, severities and milestones for defect handling. Debugging processes need to emphasise classification and prioritization before solving similar defects, like a triage, which is an imperfect art, where a certain amount of inconsistency is inevitable (Villa, 2003, p.9).

Maintainer

Michlmayr and Hill (2006) suggest an organisational improvement for daily up-keeping of the software, such as bug follow-up. The implementation of a backup maintainer could spread the load across several maintainers, which improves reliability and shortens defect removal time (Michlmayr and Senyard, 2006, p.145). The change from an individual to a team of maintainers leads to an unavoidable increase in communication complexity and requires mechanisms to increase the effectiveness before implementation.

Security Critical Defects

Some developers argue that OSS has a higher potential to achieve greater quality and can react faster to critical defects, such as security bugs (Michlmayr *et al.*, 2005). However defect handlings for commercial or security critical application face quality problems regarding reliability and response time. Security critical issues especially require a reliable and quick solution to prevent further complications. Michlmayr *et al.* (2005) observe that security updates are made in timely manner but sometimes fixes are not available. Due to the voluntary nature of OSS projects, defects are solved immediately by developers, if it serves their own interests.

Tuma (2005) argues that security of OSS is less safe, especially when the software is deployed in security critical environment, such as defence systems. But OSS is not automatically more or less secure than proprietary software, as within both development approaches misuse cannot necessarily be prevented (Wheeler 2004). Moreover, advocates of the OSSD see a benefit in the approach, since “it permits anyone to perform an independent review” (Tuma 2005). The applicability of “code reviews” performed by the large development community reduces the defect probability and contributes to increased system security (Wieland 2004). Thus, the security of OSS benefits from the collaborative development approach.

3.5.8 Infrastructure Characteristics

Infrastructure plays an important role in a dispersed and collaborative environment to support project communication and cooperation effectively (Halloran and Scherlis 2002). Michlmayr *et al.* (2005) argue that the different use of project infrastructure may have im-

portant implications for project quality. This observation needs to be set in relation to the project size, because large projects tend to make more use of tools (Zhao and Elbaum 2000). Halloran and Scherlis (2002, p.2) find that collaboration tools are widely used, including web portals, source code control, code viewers, mailings list, bug tracking systems, testing tools and instant messaging. Zhao and Elbaum (2000) observe that a surprisingly small number of people use testing tools in OSS projects and half of the projects do not take advantage of the coverage concept of tools. Some projects have strict policies on the automatic build systems, halting further development until the defect is resolved (Michlmayr *et al.*, 2005). Villa (2003) concludes that effective tool usage requires tight integration with the development process and emphasises the relevance of infrastructure topics, such as bug tracking system, version control system, automatic builds or mailing. Michlmayr (2005) analyses the links between process maturity and project success and finds that OSS projects benefit from the use of mature processes. The study shows that projects using version control tools, effective communication mechanisms and varied testing strategies seem to be more successful (Michlmayr 2005).

Specific Tool Usage

Zhao and Elbaum (2003) notice a high usage of configuration management tools and bug-tracking tools, which increases with the maturity or the project size, as large projects tend to make more use of development tools. The use of bug tracking systems has the advantage of extending the peer review process (Michlmayr and Senyard, 2006, p.134). A professional use of bug tracking tools requires triage rules (Villa 2003), such as clustering and prioritization. However, the integration of a bug-tracking system does not only enables efficient tracking and prioritization of defects but also "promotes user involvement by encouraging a new class of volunteers to join the project" (Michlmayr and Senyard, 2006, p.134). Thus, the implementation of tools supports the collaborative development processes and contributes to high quality (Aberdour 2007).

3.6 Summary and Conclusion

The basic idea behind the OSS concept comprises the availability of the source code, the right to modify derived work, the free redistribution, the integrity of the author's source code and no restrictions for licenses (OSI 2005). The OSSD model can be characterized as a collaborative "bug-driven" development of globally distributed voluntary participants with very diverse capabilities and qualifications (Dietze 2005). The interaction occurs exclusively through web-based technologies and development activities are executed in parallel, frequently delivering new software releases. The OSSD model differs from traditional plan-

driven approaches. While traditional methods have defined teams and requirements, the OSSD follows an iterative and parallel development approach with a user driven development direction, no central management, free participation, large development communities and effective user testing. Under the OSSD model several aspects affect quality management. For example the development methodology is often not documented, testing and QA methods are informally applied, projects do not collate empirical evidence regarding quality and only few measurable quality goals are defined (Aberdour 2007).

Quality Assurance Aspects

The OSSD model delivers successful products that seem to be high quality, such as Linux or the Apache Web Server. Hence, certain practices to assure software quality may exist. The success of the OSSD model in delivering superior products makes it important to explore further this phenomenon. The studies by Zhao and Elbaum (2000, 2003), Halloran and Scherlis (2002), Koru and Tian (2004) or Michlmayr (2005) confirm the uniqueness of the OSS model and provide evidence for Raymond's lifecycle. Their researches show that user participation is extremely high, defect-handling processes follow mainly structured approaches, testing takes a significant portion of the software life cycle and there is a high usage of configuration and bug tracking tools. However, project documentation is often rare, design documents are lacking, source code may remain unmaintained when developers leave the project and testing faces complex issues where developers may have limited access to diverse platform configurations. Successful projects make more use of version control tools, systematic testing and effective communication through the deployment of mailing lists (Michlmayr 2005). The author agrees with Aberdour (2007) that QA under the OSSD depends on large sustainable communities, effective debugging, code reviews, high modularity and rapid release cycles. Moreover, "the project environment and culture are as important as system design when creating high-quality software, but success depends on a highly organized approach, with sophisticated tool support for collaboration, debugging, and code submission" (Aberdour 2007).

Recent Findings

Recent studies provide useful information and constitute the basis for this research. Certain quality issues exist, such as the risk of missing or unspecified design documents, the handling of unsupported code or the impact of release management strategy on quality. Testing processes face challenges to cover all possible combinations. The bug reporting quality affects the efficiency of the defect-handling process. The availability of documentation affects the product quality and subsequent processes, such as knowledge capturing and transfer. Communication and coordination processes may explain how large projects handle the com-

plexity and how they benefit their community efficiently. The impact of the voluntary nature of participation needs to be analysed and how knowledge is captured.

The research results cover a range of QA practices, observed in various target groups. Some of the results may be outdated due to enhancements in the OSSD model, as it is constantly altering with the growing interest in the market. An empirical view of applied QA practices in conjunction with project success measures is missing. A broad study of applied QA aspects is required to gather the actual data within the target groups of mid- to large sized projects.

Research Focus

From the literature survey, it is clear that general factors of OSS projects need to be explored to provide a classification in terms of size, application types and project structure. Further investigation of the people involved may provide evidence on how OSS projects benefit from their communities regarding the participants' experiences, their conducted roles and their underlying motivation. The exploration of the development activities that impact quality assurance aspects are of major interest, such as code reusability, code modularity, handling of abandoned code, average code change between releases and the release strategy itself. A further investigation into testing processes may show the efficiency of user testing, peer-reviews and quality checks on the software quality. An analysis of defect handling processes may clarify their quality and contribution to an efficient collaborative development process. Furthermore, the analysis of communication and coordination practices could provide evidence about the effectiveness of the collaborative development approach. An analysis of the documentation may explain how projects capture their knowledge and support team integration. The exploration of tool support under the OSSD model could provide evidence about the adaption of those predefined processes. Therefore the research will explore applied QA practices with a focus on general characteristics, the design and development processes, the testing and defect handling practices, the extent of review and inspections processes, the documentation approaches and infrastructure aspects. Furthermore, an investigation into mature projects could deliver evidence of applied key practices.

Further Research Direction

The assumption is made that the applicability of successful QA practices, methods and guidelines can contribute to increasing software quality in OSS projects. Some advocates of the OSSD movement argue that OS projects have a free nature and need no strictly limited and pre-designed processes, because this would decrease the developer's motivation. However many larger projects show processes similar to traditional engineering projects but with less formality and organisational structures. As the OSSD model delivers many successful

products, it can be argued that QA practices are applied to a different extent but show similarities to traditional engineering approaches. Further research of mature OSS projects is undertaken to verify the quality assurance processes and to identify key practices. These findings contribute to the establishment of a quality model for OSS.

Part 2 - Research

The second part of the thesis discusses the research strategy, the survey research and the statistical analysis.

The research paradigms are examined and their underlying ontological, epistemological, ethical aspects and methodologies for research in Information Systems are selected. The use of nomothetic and ideographic methodologies in Information Systems are discussed and an appropriate research model is selected.

The survey research method is chosen and its aims, objectives and general assumptions are explained. Based on the research questions the design of the questionnaire is developed and the appropriate target group is selected. The survey execution and the first observations are critically discussed.

The statistical analysis is performed using SPSS and the research questions are critically examined. The survey findings are discussed and the chapter closes with the investigation of the key elements that contribute to QA of the OSSD.

4 Research Strategy

This chapter discusses the research strategy and methodology. Ontological and epistemological assumptions are explored and the underlying research methodology is introduced, suggesting a chain of methods. Each research method is explained, followed by an outline of the techniques used for data collection.

4.1 Research Paradigm

The research paradigm "consists of assumptions about knowledge and how to acquire it, and about the physical and social world" (Hirschheim and Klein 1989). In science the two assumptions are distinguished, as those that are associated with the way knowledge is acquired (epistemological assumptions) and those that relate to the view of the social and technical world (ontological assumptions). Burrell and Morgan (1979) propose four paradigms for the analysis of social theory. Hirschheim and Klein (1989) adapt the framework from Burrell and Morgan and classify four approaches to Information System development, such as functionalism, social realism, radical structuralism and neo-humanism (see figure 20).

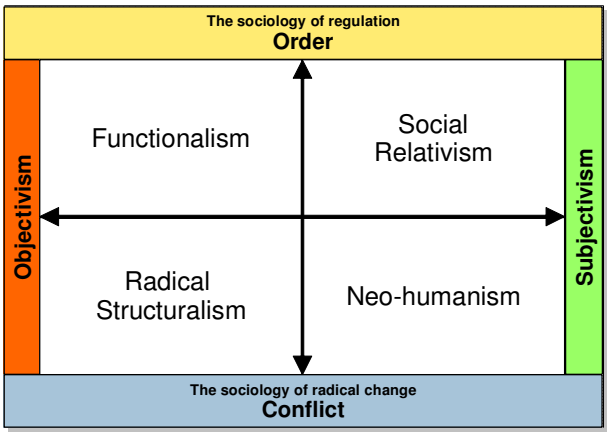

Figure 20. Information System Development Paradigms adapted from Hirschheim and Klein (1989)

4.2 Research Assumptions

According to Iivari *et al.* (1998), paradigm assumptions are divided into ontology (assumptions about the nature of information), epistemology (assumptions about human knowledge and how it can be achieved), research methodology (assumptions about the preferred research methods) and ethics (assumptions about the implied values of the research), as depicted in figure 21.

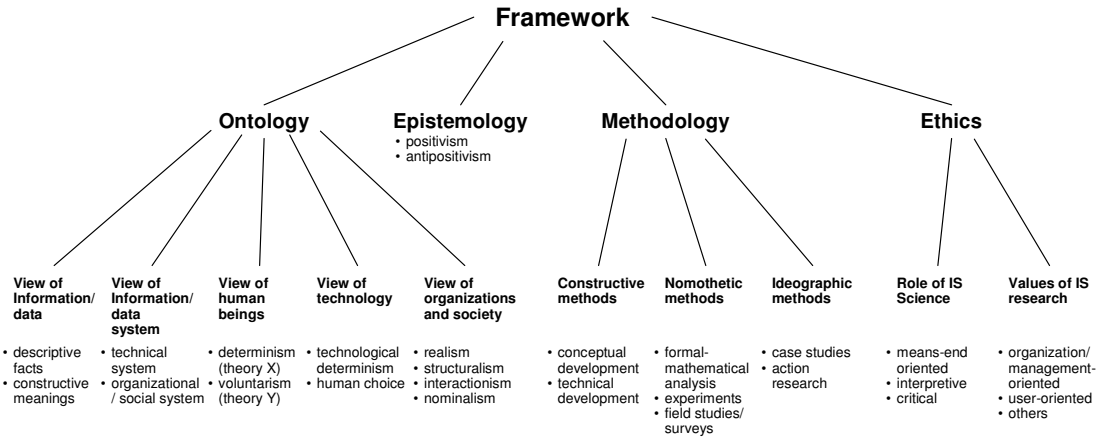

Figure 21. Framework for Paradigmatic Analysis (Iivari *et al.*, 1998)

4.2.1 Ontology

Ontology refers to the structure and properties of "what is assumed to be the nature of the information systems (IS)" (Iivari *et al*., 1998, p.164). With the ontological assumption, researchers define their view of the social and technical world (Hirschheim and Klein 1989). Burrell and Morgan (1979) define the dimensions of the ontological choice: a subjective-objective viewpoint and an order-conflict viewpoint. An objective view of reality follows the assumption that measurements are taken independent of the observer. The subjective view of reality assumes that each individual has a unique view of the world. The order-conflict dimension is obtained from the sociological assumption about nature of society. The view of society differentiates between stable and open to consensus, which reflects the "order" dimension and a "conflict" view of society that stresses change and disintegration (Hirschheim and Klein, 1989, p.1201).

The ontological assumption of this research will adopt the neo-humanist approach, which has a subjective view, stressing change, conflict and "reflects the desire to improve the existence of organisational actions by developing information systems that support a rational discourse" (Hirschheim and Klein, 1989, p.1209). The ontology adopted follows realism for technical interests and nominalism for "mutual understanding and emancipation of interests" (Hirschheim and Klein, 1989, p.1209). Realism hypothesises that the social world exists independently of the individual and is made up of relatively immutable structures. The nominalism view postulates that the social world is not real and made up of an artificially created concept, which is used to structure reality (Burrell and Morgan, 1979, p.4).

Iivari *et al.* (1998) propose that the ontology of IS research is concerned with the phenomena of information and data, information systems, human beings, technology, and human organi-

sations and society. The author's assumptions are based on the proposed ontology by Iivari *et al*. (1998).

View of Information and Data

Information and data are viewed as descriptive facts rather than having constitutive meaning. "Computer systems model a part of the reality surrounding them" and "reflect an interpretation of reality" argues Andersen *et al.* (1990, p.212) that does not suggest that IS may influence the "process of constructing an interpretation of reality" (Iivari *et al.*, 1998, p.184).

View of Information System

The view of an Information System is that of a technical and social system. This research considers the view of Information Systems as "technical systems with social implications" rather than "social systems only technically implemented" (Iivari, 1991, p.256).

View of human beings

Burrell and Morgan (1979, p.6) classify human beings as either deterministic or voluntary. The determinist view regards a person's activities as completely determined by the environment in which they are located. The voluntarist view assumes that a person is completely autonomous and self-directing. The human beings view includes both deterministic and voluntary elements, hence it is assumed that persons control their activities within the boundaries of the social structure they are affecting through their actions.

View of technology

Technology is seen as "designable and malleable by professional design choice" (Iivari *et al.*, 1998, p.184). It is assumed that technology is governed by human choice, where persons are "responsible for their development and consequences" (Iivari, 1991, p.256)

View of organisations and society

Burrell and Morgan use the dimension of realism versus nominalism to describe the ontological assumptions regarding social reality. A less radical position can be found by Iivari (1991, p.256) who describes the view of organisations and society by structuralism versus interactionism. Structuralism covers a "formal-ratio" and "structural" view of organisations. Interactionism is used in a broader sense, covering "interactionist" and "political" points of view and emphasising organisational processes as determinants of the organisational phenomena (Iivari *et al.*, 1998, p.173). The underlying organisational view is structuralist with some interactionist elements.

4.2.2 Epistemology

Epistemological assumptions "are concerned with the nature of knowledge and the proper methods of inquiry" (Iivari *et al.*, 1998, p.174). Burrell and Morgan (1979) classify the two opposite viewpoints of epistemology into positivism and anti-positivism. The "positivist" seeks to explain "what happens in the social world by searching for regularities and causal relationships between its constituent elements" (Burrell and Morgan, 1979, p.5). Anti-positivism, conversely, maintains that "the social world is essentially relativistic and can only be understood from the point of view of individuals who are directly involved in the activities which are to be studied. Anti-positivist rejects the standpoint of the "observer, which characterises positivist epistemology as a valid vantage point for understanding human activities" (Burrell and Morgan, 1979, p.5).

This research uses an anti-positivist assumption, which maintains "one can only understand by occupying the frame of reference of the participant" (Burrell and Morgan, 1997, p.5) and assumes that the researcher views the topic "from the inside rather than the outside" as participant in the action. Anti-positivism "emphasises human interpretation and understanding as constituents of scientific knowledge" (Iivari *et al.*, 1998, p.174).

4.2.3 Ethics

"Ethics of research refers to assumptions about the responsibility of a researcher for the consequences of his/her research approach and its results" (Iivari *et al*., 1998, p.175). Research with Open Source software development raises the same ethical issues as other disciplines such as literary or policy in which human beings are object of analysis (Vinson and Singer 2001). Software developers can be harmed by the research, as questions can cause offence regarding e.g. gender, ethnicity bias or performance criticism. As El-Eman (in Vinson and Singer 2001) noted, analysis could rank the programmers according to the defects rate of their code, thus adversely affecting the careers of the worst programmers. It can be argued that this is acceptable from an ethical perspective, in the way that good programmers are rewarded. The author agrees with Vinson and Singer in the fact that a kind of metrics scale could not capture the programmer's value. The number of defects says nothing about the quality because criteria such as the effort or the possibility to correct defects, the level of difficulty, maintainability and reusability have to be analysed. A developer's source code that contains more bugs but are ones that could be easily fixed is worth more than another developer's source code with less defects but which requires high effort to correct. Therefore, the defect rate does not adequately measure programming skills. Vinson and Singer pointed out that a metrics-based ranking of programmers can be misleading, resulting in

adverse judgements being made of the programmer's true worth. They think that the potential for harm increases the importance of obtaining informed consent. This means if the potential for harm is eliminated, the need for consent would be greatly reduced.

If personal identifiers are removed from the analysis data, anonymity can be ensured. Although confidentiality for individuals is guaranteed, the offensiveness or harmfulness of the final research results must be considered. Within this research project a nomothetic and ideographic analysis of quality criteria of different Open Source projects is conducted. This analysis provides information about relevant quality issues. In order not to do harm to any participant or organisation by the research results, all personal identifiers are eliminated and the data are presented anonymously. These measures ensure that extracting individual identities is simply not possible and the ethical aspects are maintained.

4.2.4 Methodology

"Research methodologies refer to the procedure used to acquire knowledge about IS development approaches and related IS development methods and tools" (Iivari *et al.*, 1998, p.174). Burrell and Morgan (1979) distinguish between ideographic (qualitative techniques) and nomothetic (quantitative techniques) research methods.

Nomothetic methodologies base research upon systematic technique and "focus upon the process of testing hypotheses in accordance with the canons of scientific rigor". Ideographic methods emphasise the "analysis of the subjective accounts which one generates by "getting inside" situations and involving oneself in the everyday flow of life" (Burrell and Morgan, 1979, p.6). The authors identify a third research methodology described as constructive method that is concerned with the engineering of artefacts, which are either conceptual or technical.

The research will adopt both, nomothetic and ideographic methods as explained in the following section. The nomothetic method is used to obtain quantitative data through a field study. Ideographic methods are applied to "place considerable stress upon getting close to one's subject and exploring its detailed background and life-history" (Burrell and Morgan, 1979, p.6). According to Iivari *et al.* (1998, p.175) ideographic methods appear closely associated with the idealist ontology position in IS.

4.3 Research Methods

The research method is the "strategy of inquiry which moves from the underlying philosophical assumption to research design and data collection" (Myers 1997). The selection of the appropriate research methods consequently influences the way data are collected and implies certain assumptions. Quantitative research is carried out under the positivist tradition, as interpretative and critical positions are not meaningful (Straub *et al.,* 2005). Qualitative research methods are used when anti-positivist research paradigms, such as interpretive and critical, are applied. The epistemological assumptions for qualitative and quantitative research are shown in figure 22.

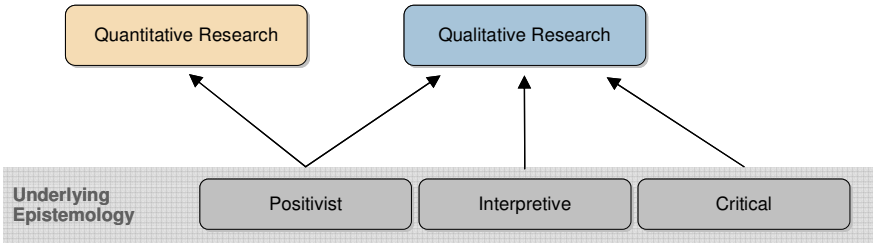

Figure 22. Epistemological Assumptions (Straub *et al.*, 2005)

4.3.1 Survey Research

The survey method refers to quantitative analysis and aims to collect a verbal or written response to questions or statements (Emory 1980, in Straub *et al.*, 2005), (Straub 1989), (Kraemer 1991), (Kraemer and Dutton 1991), (Pinsonneault and Kraemer 1993), (Newsted *et al.*, 1998). Surveys can be effective in gathering data about individual preferences, expectations, past events, and private behaviours. Emory, 1980 (in Straub *et al.*, 2005) points out that the versatility of this method is its greatest strength and argues that this method is the only practical way to gather many types of information and the most economic way in many other situations.

According to Kerlinger in Malhotra and Grover (1998, p.409) two major types of survey research are to be distinguished. The first type is classified as "exploratory" and aims to become more familiar with the topic. In this case, no model exists and the aim is a better understanding of the research topic. The second type is described as "descriptive" survey, which is an indispensable approach to study phenomenon at an early stage of the research, as it develops the units that comprise theories (Dubin in Malhotra and Grover, 1998, p.409). This type aims to describe the distribution of phenomena in a population, thereby ascertaining facts.

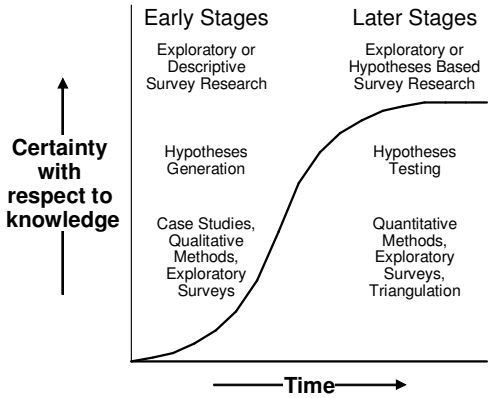

Figure 23. Research Maturity Cycle adopted from Malhotra and Grover (1998)

In an early stage of the research (see figure 23) exploratory and descriptive surveys help to identify the concepts and the basis for measurement (Malhotra and Grover, 1998, p.409). In later mature stages of research, Malhotra and Grover suggest using explanatory surveys, where “how” and “why” variables can be effectively measured and relationships among them can be studied. Hypotheses could be basic, such as showing relationships between variables, or directional, i.e. positive or negative.

This research uses an exploratory survey approach, with the aim of gathering a further understanding about the object of study and to verify the existing theoretical knowledge. With this approach, a representative sample of organisations is studied in order to identify common relationships across them and to provide generalisable statements about the object of study (Gable, 1994, p.114). Vidich and Shapiro (1955, pp.31) state, "without the survey data, the observer could only make reasonable guesses about his area of ignorance in the effort to reduce bias." This method provides important insights into the object of study but cannot be used “to objectively verifying hypotheses”, argues Garble (1994, p.114). The method has certain drawbacks. Once the survey is being executed, the researcher cannot take many countermeasures, when crucial items are omitted or respondents misunderstand the questions. Furthermore, the method provides only a “snapshot” of the actual situation and does not allow understanding of the underlying meaning of gathered data. Therefore, the survey research is undertaken using an interpretative tradition using a social relativism paradigm.

4.3.2 Action Research

Action research methods are applied where the researcher attempts to “obtain practical results of value to groups with whom the researcher has allied him/herself while at the same time adding to the body of theoretical knowledge” (Galliers, 1985, p.282). Hence, action research aims to contribute to practical knowledge of participants as well as scientific knowledge, due to a “joint collaboration within a mutually acceptable ethical framework”

(Rapoport, 1970, p.499). In addition, there should be a mutually satisfactory outcome between the researcher's theoretical interests and the organisation. The researcher carries out the research as participant and not as observer. Hence, this method can be characterised as anti-positivism and interpretative or critical research.

According to Susman and Evered (1978), action research follows a five phase cyclical process, such as diagnosing, action planning, action taking, evaluating and specifying learning. The diagnosing phase focuses on the identification of the primary problem and aims to develop a working hypothesis about the nature of the organisation and its problem domain (Baskerville 1999). Researchers and practitioners collaborate in the action planning, which is guided by the theoretical framework, in order to solve these primary problems. In the action taking phase, the researchers and practitioners implement collaboratively the planned actions with direct or indirect application of an instructional strategy or technique. The evaluation phase refers to the assessment of the instructional strategy or technique. The activity "specifying learning" is formally undertaken as an ongoing process to acquire new knowledge.

The underlying problem of the action research method is that "it cannot be wholly planned and directed down a particular path" (Checkland, 1981, p.153). Checkland concludes the researcher might express his own aims, but is not able to implement them with certainty in the experiment. Hence, the researcher must direct his research according to where the situation leads him.

4.3.3 Case Study

Case study research is a qualitative method that is used in IS. Craig Smith (1990) describes case studies as a "way of organising social data so as to preserve its unitary character". Case studies are empirical inquiries that investigate phenomenon within its real-life context, when the boundaries between phenomenon and context are not clearly evident (Yin 2002). The case study research can use a positivist, interpretive or critical epistemology, depending on the underlying research assumption. Yin (1993) supports a positivist definition and argues that evidence should link up research questions arising from rival theories. Similar thoughts can be found by Benbasat *et al.* (1987), who emphasise the importance of testing hypotheses when case studies are performed. Walsham (1993, 1995b) considers the views of Benbasat and Yin as positivist but agrees that the approach emphasises "how" and "why" questions that indicate an interpretative approach. According to Benbasat *et al.* (1987, p.370) case studies are a well-suited approach to capture knowledge of practitioners and are a useful approach when the phenomenon needs to be explored in its natural setting. A case study

method is an "appropriate way to research an area in which few previous studies have been carried out" (Benbasat *et al.*, 1987, p.370), which applies to this research.

The non-representativeness and a lack of statistical generalisability are criticisms made of the case study approach in the literature. Miles and Huberman (1994) see a lack of a step-by step data analysis and argue that case studies cannot provide generalisability in the statistical sense. Pettigrew (1985) opposes this view, arguing that case studies are useful in developing and refining generalisable concepts and adds that multiple case studies can lead to generalisations regarding propositions. The validity of the case study approach derived from an interpretative epistemological stance, which is based on the "plausibility and cogency of the logical reasoning applied in describing and presenting the results from the cases and in drawing conclusions from them" (Walsham, 1993, p.15).

4.3.4 Triangulation

The technique of triangulation is broadly defined by Denzin, 1978 (in Jick 1979) as "the combination of methods in the study of the same phenomenon". Denzin distinguishes two methods, the "within-method", which uses multiple techniques within a given method to collect and interpret data, while "between-methods" triangulation tests the degree of external validity (Jick, 1979, p.603). Jick says triangulation "can capture a more complex, holistic and contextual portrayal of the units under study. The use of multiple measures may also uncover some unique variance which otherwise may have been neglected by single methods." Triangulation allows similar phenomena to be examined from multiple perspectives and enriches the understanding by allowing new or deeper dimensions to emerge. The effectiveness of triangulation derives from the compensation of weaknesses of each method with the strength of another. "Triangulation purports to exploit the assets and neutralize, rather than compound, the liabilities" (Jick, 1979, p.604). Within this research both "within" and "between-methods" are used. The "between-methods" are accomplished by using a survey and a case study in this research.

4.4 Choosing the Appropriate Research Methodology

Selecting the appropriate research methods in Information Systems is itself an area of debate and needs to be carefully chosen (Galliers and Land, 1988 in Jarvenpaa 1988). Choosing the research approach means considering the research requirements and proposing suitable methods for the area of study. Vitalari (1985) concludes that research studying the impact of IT and IS on organisations may utilise the survey and case study method. Therefore, a nomothetic approach with data collection using the survey method and an ideographic approach

with case studies are considered as main elements. The action research method differs from the other methods from the viewpoint of investigation, where the researcher is actively association himself with particular practical outcomes of the research (Galliers 1981). The method demands the involvement of the researcher in the object of study, which assumes theoretical knowledge in the specific area. This method is not applied within this research, as its direction cannot be fully planned and it does not assure an objective viewpoint. "The action researcher is not an independent observer, but becomes a participant, and the process of change becomes the subject of research" (Benbasat *et al.*, 1987, p.371).

This research adopts the methodology introduced by Galliers and Land, (in Jarvenpaa 1988) and follows the suggested chain of methods, as discussed in chapter 1.4:

- Research Question
- Survey Research
- Theory Building
- Case Study
- Theory Extension

The chain of methods is adjusted accordingly as outlined in figure 24. Theory testing in the field is not applied, as an interpretative approach rather than a statistical evaluation is undertaken to evaluate the findings. Therefore, the case study research is selected to validate the theory in practical projects. Within this method, quantitative and qualitative data collection methods are combined.

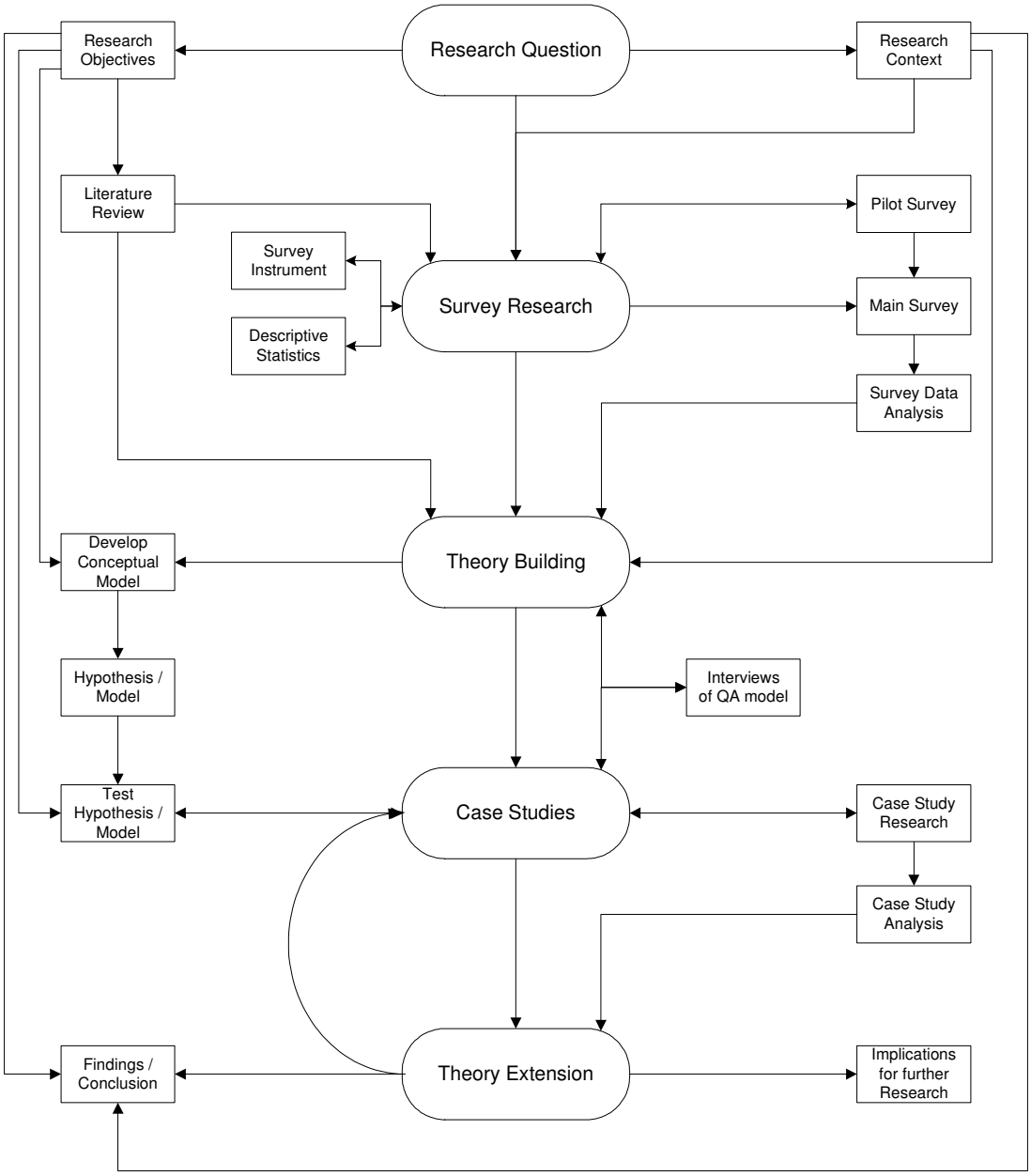

Figure 24. Proposed Research Approach

Research Questions

The research questions are stated to define the overall structure of the thesis and are subsequently refined during research.

Survey Research

The survey research method is used to obtain empirical data and a greater understanding about applied Quality Assurance practices in the OS domain. The purpose is to learn from the experience of the surveyed target group and to validate the findings of the literature review. In summary, it addresses professionals in the OS development area, mainly developers who contribute to the development process. The survey method is primarily chosen to get

access to the OS communities and is seen as an appropriate approach to reach the participants in different time zones. The method makes use of the collaboration method of a web portal and distribution lists, which are common in the OS environment. An electronic survey with direct mailing is used to ensure online access to the questionnaire. An appropriate target group is chosen according to pre-selected project criteria, such as topic, status, size, as well as personal factors, emphasising a project role with development expertise. Therefore, statistical data from large OS web portals are analysed to identify a proper target group. The target group comprises all persons, who are assigned to the selected group of projects. Within the target group a further reduction according to the personal role and project is performed, to obtain a broad distribution list containing at least one or two persons per project. Although the survey uses a positivist methodology, the results of the survey are interpreted using a social relativism paradigm, as discussed in chapter 4.3.1.

Theory Building

The survey provides further background information and empirical evidence of QA practices in the OSSD model. Furthermore, it offers insights into a large range of projects, which might be useful for further research. The survey results contribute to the understanding of applied QA practices and help to identify key processes that could be incorporated into the QA framework. Several interviews with subject experts are conducted to obtain further knowledge and to verify the framework development. The findings from the interviews contribute to the theory building and the subsequent case studies.

Case Studies / Theory Testing

The case study method is used for theory testing of the developed framework. The case study research attempts to describe the relationships that exist in reality (Galliers 1985). Consequently, an external validation with current OSS projects is required. Within the method, the research is undertaken from the viewpoint of an independent observer across several organisations. The researcher does not associate himself with the organisations in order to keep an objective position. Several case studies are conducted in selected OS projects in order to verify the framework model. Triangulation is deployed to gather the convergence of qualitative and quantitative results.

Theory Extension

The case study findings deliver further information about the applicability of the theory, the model completeness or other factors that need to be considered. These outcomes may lead to theory extension where there is incompleteness of the model or lack of theory, which subsequently results in research iterations and may require a re-validation of the model. This itera-

tion finishes when the research outcomes fulfil the expected results. A final conclusion is drawn and the findings and their contribution to knowledge are explained.

4.5 Summary

The epistemological assumption of this research is anti-positivist. The ontological assumption of this research adopts the neo-humanist assumption, which has a subjective view, stressing conflict.

Following the framework of Iivari *et al.* (1998), ontological assumptions of information and data are viewed as descriptive facts. The view of an Information System is seen as "technical systems with social implications" (Iivari, 1991, p.256). The view of human beings is seen as voluntary within the bounds of a social structure. It is assumed that technology is governed by human choice, and that organisations are structuralist with some interactionist elements. Ethical assumptions are maintained so as not to harm developers and anonymity of results is ensured by the elimination of personal identifiers.

The research adopts the framework of Galliers and Land (1988), which suggests a chain of methods. A nomothetic approach using the survey research is selected to gather data from the field. The survey method is applied according to the approach of Malhotra and Grover, which is explained in section 55.2. Although the survey method has been criticised in the literature (Kraemer and Dutton, 1991, p.3), it can provide useful results when combined with qualitative data through triangulation. Therefore, open-ended questions have been integrated into the questionnaire (see section 5.4) to gather qualitative information. Chapter 5 describes the applied survey method within this research, while the survey findings and its contribution to knowledge is shown in chapter 6.

Based on findings from survey and literature, the key processes of the QA framework are established and refined. The development of a process and measurement approach and its collation into the QA framework is described in chapter 7.

Multiple case studies are conducted, in order to validate the knowledge and to contribute to theory testing. The case study research approach and the findings are explained in chapter 8.

5 Survey Research

In this chapter, the aims and objectives of the survey research are described and the underlying assumptions of the approach are explained. Based on the findings of the literature review, different criteria affecting software quality under the OSSD are analysed. These criteria are used to build up the questionnaire. The rationale behind the questions and the correlations are explained. The preliminary questionnaire is refined after consultation with subject experts and adjusted accordingly. The definition of the target group as well as the selection process is explained. The chapter closes with a conclusion about the selected research approach.

5.1 Research Questions

As stated in chapter 1.1 the first research question explored in this dissertation is:

> *How is Software Quality Assurance in OSS projects applied and what key practices characterise mature projects?*

According to the findings from the literature review, the following areas are of interest for further research, such as: general and organisational issues, personal issues, software engineering processes, QA approaches, testing and defect handling, documentation and tools. More detailed questions which arise from these, are:

- What are the general factors of OSS projects and how are they classified?
- Who are the persons involved, what roles do they play and what is their motivation?
- How are OSS projects structured and what proportion does each phase typically have?
- How are Quality Assurance processes in the design and development phase conducted?
- How is testing applied?
- What defect handling processes are applied?
- How is team integration and communication established?
- What kind of documentation is considered?
- What kind of tools are commonly applied to support the entire project lifecycle?
- What kind of Quality Assurance tasks and improvements are applied in projects?

These subsequent research questions lead to several successive issues and represent the preliminary structure for the development of the questionnaire.

5.2 Survey Research Methodology

The survey research is undertaken using an interpretative tradition and social relativism paradigm as discussed in the chapter 4.3.1. Epistemologically this method is selected to obtain and complement knowledge by “triangulation” (see chapter 4.3.4). The usefulness of the

survey method in objective or positivist research, as well as in more subjective or interpretivist research has been discussed by Newsted *et al.* (1996). An exploratory survey research is applied for the reasons discussed in chapter 4.3.1.

The findings of the literature review are used to define the measures of interest. The constructs and the expected relations are defined, which guides the investigation and allows the identification of relations and regularities in the observed data. An electronic survey method is applied. This method provides similar results to traditional paper based surveys, but can be easily conducted via the Internet (Newsted *et al.*, 1998). The author agrees with Straub (1989) who indicates that the survey instrument can bring greater clarity to the formulation and interpretation of research questions. In the process of validating an instrument, the researcher is engaged in a reality check. A validation of the questionnaire is performed to find out "how well conceptualization of problems and solutions matches with actual experience of practitioners." (Straub, 1989, p.148). The survey method is applied according to the approach of Malhotra and Grover (1998) in Newsted *et al.* (1998). Their approach is selected on the basis that the research obtains an anti-positivist epistemology and the survey results are interpreted using a social relativism paradigm. The approach can be outlined as follows:

- Determination of the unit of analysis (e.g., the individual, group, or organisation)
- Creation and use of multi-item scales
- Pre-testing and use of pilot data
- Assessment of both construct and content validity
- Assessment of reliability
- Random sampling from a defined sample frame
- Determination of an appropriate response rate and evaluation of non-response bias
- Determination of the statistical power of the final analysis

5.3 Goals and Assumptions

The survey focuses on Quality Assurance processes in the OSSD cycle. Therefore, core development processes, rather than technical issues are considered. It is assumed that the development processes follows the OSSD development cycle as described within chapter 3.4.

Within OSS projects, development methods are often blurred, as several nuances from agile to plan-driven methods can be found. The Open Source development approach is taken into account in the design of the questionnaire for the reasons discussed in chapter 3.4.3.

The target group must represent project participants with widespread knowledge about development processes in OSS projects. It is assumed that participants with large practical experience about software development methods, who actively contribute to OSS projects, can provide the most useful insights into applied practices and methods. The personal experience of the participants is of certain interest. This unit of analysis is chosen to avoid influences

from other participants, groups or organisations. Hence, individuals who have project roles such as developer, committer or project manager represent an ideal target group.

Small projects with one to three developers tend often to a "need" driven approach, which could be characterised by a "demand on request" or flexible management, which allows bridging of certain process gaps. Hence, this research focuses mainly on mid- to large size projects which consists of a developer team of up to ten or up to twenty and above developers, due to the assumption that these projects are facing higher complexity and suffer from a lack of processes and methods for the reasons stated in chapter 3.4.3.

The first goal of the survey is to obtain empirical information about applied practices and to validate theoretical knowledge as discussed in chapter 3.5. Furthermore, quality assurance practices and methods of successful projects are of interest. It is assumed that a certain threshold for a "successful project" can be determined when analysing the data. Hence, successful projects can be extracted from the sample to examine common patterns and correlation of variables. The project success measures, as defined by Crowston *et al.* (2006), are applied (as stated in chapter 3.5).

The aim of the survey is to find evidence for common processes, which contribute to quality assurance under the OSSD. Therefore, the major assumptions are incorporated into the survey design:

- Quality criteria and methods
- Project success measures

The area of quality criteria and methods aims to cover the quality problems within the OSSD and to validate the findings of previous surveys, such as Zhao and Elbaum (2000, 2003), Halloran and Scherlis (2002) and Koru and Tian (2004). Beside the validation of existing knowledge, the questionnaire is enriched with further questions, focussing on the quality problems as discussed in chapter 3.5. In addition to that, project success measures are integrated. The content of the survey comprises general statistical data, development processes, defect handling and testing techniques, documentation, infrastructure and quality issues.

5.4 Questionnaire Design

The design of the questionnaire is based upon three major assumptions. First, general project criteria are essential to classify and group the different types of projects. Second, research questions from previous studies by Zhao and Elbaum (2000, 2003), Halloran and Scherlis (2002) and Koru and Tian (2004) which focus on QA methods, tasks and processes in OSS projects are the subject of this research to obtain an actual and comparable result within the target group. Third, project success metrics as defined by Crowston *et al.* (2006) are inte-

grated into the survey to allow a first separation of projects regarding their success. The draft questionnaire is built upon the findings from the literature review. Prior researches are examined to identify and to verify the appropriate measurements.

Straub *et al.* (2004) argue that content validity is established through literature reviews and expert judges. Following this approach, a pre-test of the questionnaire is performed by interviewing practitioners in the field to assess content validity and reliability (Straub *et al.*, 2004; Malhotra and Grover 1998). A pre-test is conducted with OSS developers in the form of a pilot-survey, followed by an interview. During the pre-test, no further explanations or assistance is provided. An unstructured follow-up interview is used to discuss the findings. Based on the results of the pre-test minor modifications are made, such as changes of wording, adjustments of selected scales and removal of redundant questions.

The design of the questions follows the assumptions that the meaning of them are commonly shared, the terms are understood, the respondents have the same understanding of the investigated object and the responses are comparable for the researcher (Sudman and Bradburn 1982). In the first step, the survey instrument aims to avoid problems in the survey data that affects reliability (Fowler 1984). Second, the wording is analysed, to avoid a survey bias, which relates directly to the construct validity (Paulhus 1991). Finally, the survey data are analysed to avoid problems due to improper sampling and non-sampling errors, inconsistent administration procedures or problems with the research process itself (Sudman and Bradburn 1982).

The questionnaire was set-up as a web based survey, following Dillman's (2000) guidelines for an effective internet survey. The questionnaire starts with general questions and more specific questions are put at the end of a chapter (Dillman, 1978, pp.123-128) or in the last section of the questionnaire as recommended by Porst (1998, p.31). The question types are a combination of closed with ordered responses and open-ended to allow further explanation or answers to not anticipated scales. The questions are structured using ordinal and nominal scales. A questionnaire should not exceed 11 pages, or 125 items, which would lead to reductions in the response rate (Dillman 1978, p.55). Therefore, it comprises seven pages, containing several related questions. In total 86 questions are asked, to maintain a reasonable and manageable size. The questionnaire is thematically grouped into six main categories, designed from general to specific criteria in the following order:

- **General Issues**
  Within this part general project criteria, such as type, size, complexity as well as market availability and maturity are determined to cluster the different project types.
- **Personal Issues**
  This part records the participants role, their motivation and experience to allow a classification of the total result set.

- **Processes and Methods**
  The focus of this section is on general development criteria, such as release frequency, proportion of code changes, reusability of code or modularity. Furthermore, quality issues due to code complexity or quality improvements are surveyed.
- **Testing and Defect Handling**
  Detailed information about testing and defect handling activities, test approach, user testing, consideration of user suggestions and quality control processes are analysed. The efficiency of defect handling processes is of especial interest. This comprises process criteria, scope, security critical issues, reporting quality or follow-up procedures.
- **Organisational Issues**
  In these section organisational criteria are analysed with the focus on communication, knowledge transfer, tool usage as well as documentation approach.
- **Quality Assurance Tasks**
  The final section focuses directly on QA and improvement activities. The participant's experience of process enhancements is surveyed.

Each category comprises a set of specific research questions. The detailed specification of the research targets and their rationale is developed and summarised in the Appendix (table a.1). The finalised questionnaire and the logical structures are customised in the Online Survey Tool. The final questionnaire can be found in the Appendix, chapter A.2.

5.5 Target Group

The target group for the survey consists of individual developers contributing to mid- to large sized projects (as discussed in chapter 5.3). The source group of projects was selected from the web portal "Sourceforge", which hosts the largest amount of Open Source projects and provides additional statistical project information.

In the first step, a group of projects, which correspond to pre-selected categories, were determined. Within the group of projects, a stepwise reduction to a manageable size of 20,000 projects was performed. Therefore, large projects mainly of the following categories, such as Education, ERP/CRM/Financial, Games/Entertainment, Office/Business, Security or Software Development were pre-selected.

In the second step, for each project, two participants were chosen, who correspond to the group of developers, designers, testers or project managers. This results in a list of 6,000 potential participants, who were included in the final distribution list for direct mailing. The pre-selection of the target group turned out to be very helpful as the project information used is not up to date, which indicates certain changes within the project organisation.

5.6 Survey Execution

Newsted *et al.* (1998) discuss the approach of an electronic survey and conclude a growing acceptance of such survey methodologies. Besides the tool, the access possibilities need to correspond to the facilities and customs of the target group. A web-based survey tool allows direct access to the target group with no time or geographical restrictions. A multimode approach, which combines e-mail based communication with the participants and a web-based survey for data collection was selected. Invitations or reminders were sent as direct mailings with a personal link enclosed to the web-based questionnaire.

As platform for the survey, the Open Source tool "PHPSurveyor" was chosen, which allows a flexible customisation of the questionnaire, tracking of the results and simple statistical overview functions. The tool fulfils adequate administration functions to avoid problems with the research procedure itself (Sudman and Bradburn 1982).

The survey was executed in the period from 13th June until 10th July 2007, weekly reminders were sent out to the participants. The mailing was done in five waves with different subsets of the target group. Further details about the selected subsets of the target group and a statistical table of the responses can be found in the Appendix A.3.

Table 6. Survey Responses

Invitation Sent in Total	Failed Mailings in Total	Responses in Total	Response Rate	Invalid Records
6000	806	427	8.2%	11

The total response rate of the survey with 8.2% (as shown in table 6) is relatively low but 427 responses are a sufficient basis to allow a valid analysis. Simsek and Veiga (2000) report that electronic surveys could achieve response rates from 19.3% to 76%. Although a pre-notice was sent to all participants, many difficulties exist to retrieve valid responses. Dillman (2000) states that several potential issues may influence the response rate, such as survey length, the survey design, the contact with the participant or lack of interest. The feedback shows that in the OSS domain, some developers are overwhelmed with mailings and an aversion against interviews seems to exist. A further issue is that the survey topic and the invitation need to arouse the participant's interest, which is difficult in a non-personalised e-mail based communication. These factors are considered to be the main reasons for the low response rate.

5.7 Conclusion

The multimode approach, which combines the e-mail and web-based survey for data collection proved to be an efficient approach. The e-mail based communication with the participants as well as the administration and tracking of the results in a web-based survey tool avoided issues with the research process itself. The selected research approach follows the way of project communication in a distributed development environment, using mailing lists and allowed the survey of large number of organisations. The choice of an appropriate tool is essential, as it needs to support flexible customisation of the researcher's requirements, administrative and tracking functions and a stable user interface with intuitive handling.

Several participants give predominately positive feedback about content, topic and the survey results. Feedback from some participants' reveals issues with terms, common understanding of meaning and difficulties with the content of the questions. Participants of small projects criticize the questions regarding project organisation, which do not apply to small teams where participants are responsible for a variety of tasks. Developers using agile methods raise difficulties with questions regarding release strategy or their testing approach. For instance, new releases are built "on-request", which corresponds to a time-base approach. Testing is performed unit wise with a defined test plan. However, a detailed test plan did not exist in advance. Such projects use a structured testing approach but develop and plan their activities on request.

The survey shows difficulties in contacting the "right" person. Gathered project data are sometimes not up to date. Persons who are listed as contributors may have left the project and are now participating in other projects. Some respondents were users with no insight into the development process. In conclusion, the pre-selection of appropriate projects and participants leads to a high proportion of developers and project managers, which shows an excellent coverage of the target group. The open-ended questions in the last section of the questionnaire provide rich and detailed feedback regarding applied practices and further recommendations. The survey offers a rich amount of high quality data, which provides insights into development processes and methods in OSSD projects and is used for further statistical analysis.

6 Survey Findings and Discussion

This chapter focus on the analysis of the research questions and closes with a critical discussion of the survey findings.

6.1 Approach

The analysis process is automated whenever possible. All records were loaded into a Microsoft Excel spreadsheet to verify the data and to perform minor data conversions. Invalid records were eliminated from the result set. Out of 427, 11 responses are extracted, due to flawed or incomplete data. The main statistical analysis were done with the software package SPSS 16.0, while some minor analysis was done with Microsoft Excel. The prepared raw data was loaded into a SPSS data set. The variables are labelled, transformed when necessary and associated to their respective scales. The statistical analysis is performed in SPSS using functions such as frequencies and cross tabs statistics in order to determine the contingency coefficient or the correlation. The contingency coefficient is used for nominal data, while Pearson is used for interval scaled data and Spearman for ordinal data.

6.2 Results

The analysis of the survey results follows the structure of the questionnaire. As defined in Appendix A.2, each research question is assigned an identifier, such as “G1”. These identifiers are used to mark the subsequent analysis of each research question.

6.2.1 Project Classification

The general criteria describe the projects according to their application type, project and community size, market availability and product maturity.

The application type (G2) allows a thematic clustering of the projects. The majority of projects belong to categories: “Internet” (17.8%), “Software Development”, such as tool or system development (20.7%) or “Others” (18%), which groups minorities, such as research projects, artificial intelligence, library systems, simulations, etc. All other application types are almost equally represented with a proportion of around 5%, as shown in figure 25.

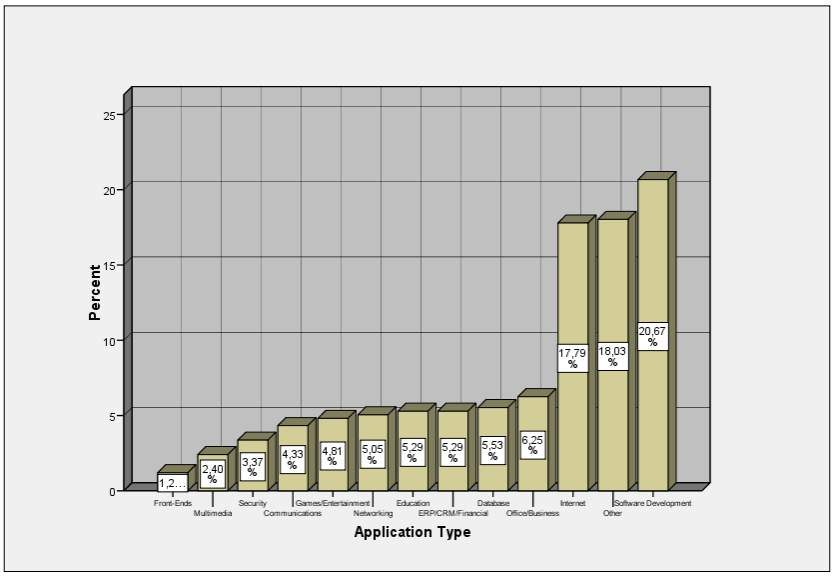

Figure 25. Projects by Application Type

The project size (G3) is determined in the lines of code (LOC) and indicates the project complexity. The group of projects is normally distributed (σ = 1.016). This criterion is used to cluster the projects into four general categories as shown in table 7. The vast majority of projects fall into the medium category, while small and large projects have each a proportion of roughly a quarter. Only 10.6% of the respondents could not classify their project size for any reason. These results show a high coverage of the target group.

Table 7. Project Sizes

	Category LOC		Frequency	Percent	Valid Percent	Accumulated Percent
Valid	Mini	<1,000	13	3.1	3.1	3.1
	Small	1,000-10,000	109	26.2	26.2	29.3
	Medium	10,000-100,000	153	36.8	36.8	66.1
	Large	>100,000	97	23.3	23.3	89.4
		I don't know	44	10.6	10.6	100.0
		Total	416	100.0	100.0	

The examination of the development language (G4) shows a high applicability of JAVA (28.1%), followed by PHP (20.1%), C (14.9%) and C++ (12.9%). The high proportion of Java and PHP may result from the sample of application types. A comparison with the application type shows that 8.1% of the “Software Development” projects are developed in JAVA and 9.1% of the “Internet” applications are developed in PHP. The majority of the observed projects use established development languages, as depicted in figure 26. The high prevalence of mature languages may offer a larger variety of established tools and supports code reusability.

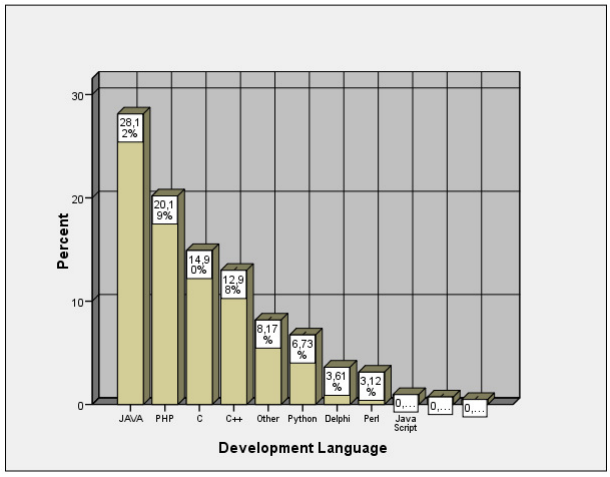

Figure 26. Development Language

The analysis of the developer team sizes (G5), see figure 27, show that the majority of projects (55.8%) are developed by small groups of 2-5 core developers, 10.3% have only one developer, 17.8% of the sample have groups of 6-10 developers and 8.7 % have more than twenty developers. This study observes much larger development groups compared to Zhao and Elbaum (2003), where only 5% of the projects had more than five developers.

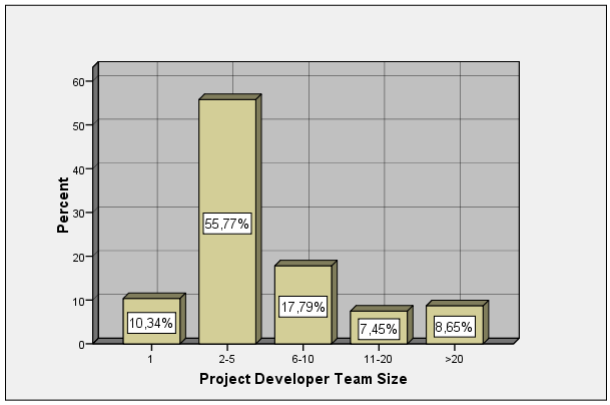

Figure 27. Developer Team Size

A comparison of developer team size to the project size shows a significant weak positive correlation (r_{SP} = .328, p < .001). Around 37.1% of the large projects have developer groups of more than ten developers. Groups of 2-5 developers mainly develop mini to medium sized projects, as depicted in figure 28.

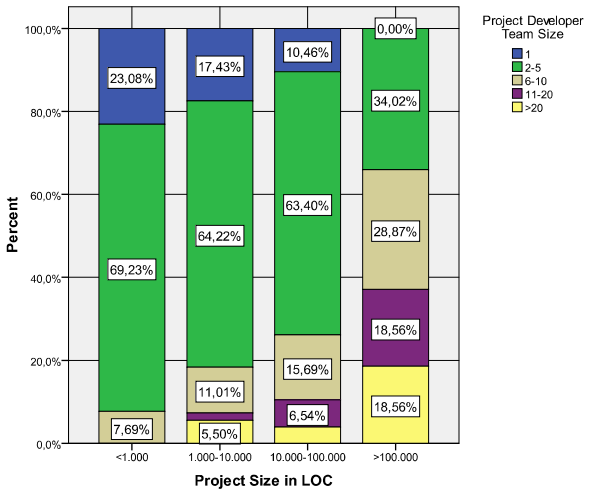

Figure 28. Developer Team Size by Project Size

The examination of the community size (G6) shows the existence of large communities. Only 26.2% responded at having less than ten users, 33.4% have 10-50 users, 9.1% have 50-100 users and 31.2% have more than 100 users. These figures are based upon the participant's evaluation and represent only estimations. Compared to Zhao and Elbaum (2003), who observed 59% of the projects have user groups of more than 50 people, this study shows smaller user groups where only 40.3% of the projects have user groups of more than 50 people. The comparison of community size to project size shows a growth of the community with project size (r_{SP} = .200, p < .001). Eighty-six and a half percent of the mini projects have less than 50 users. The proportion of medium (51-100) to large (>100) community sizes has a significant growth from small to larger projects. For a better comparability of each group, the percentage values are grouped by each category of the x-axis in figure 29.

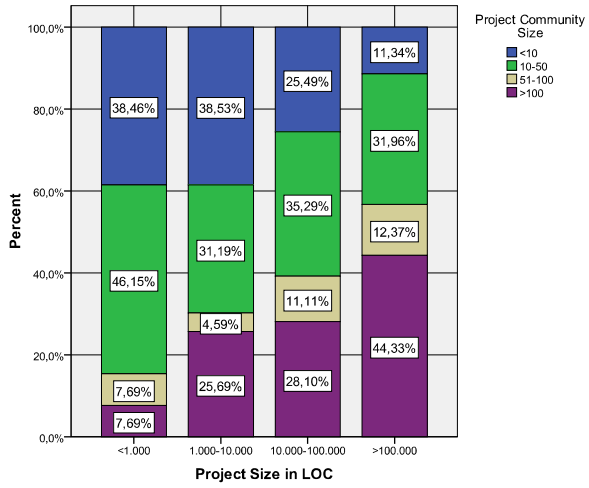

Figure 29. Community Size by Project Size

The analysis of developer team size grouped by community size in percent shows a linear growth between the variables (r_{SP} = .455, p < .001), as depicted in figure 30. Thus, the community size increases with the developer team size. That shows the support of the development teams by their communities, which constitutes an important foundation for the OSSD model.

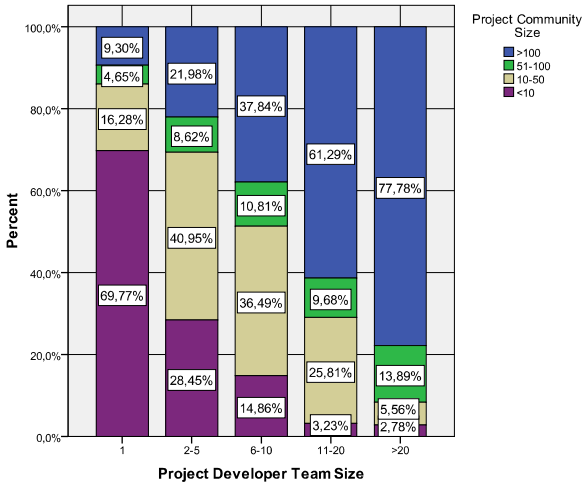

Figure 30. Community Size to Developer Team Size

Almost half the projects are more than four years in the market (G7). A comparison of the market availability with the project size shows a weak positive correlation (r_{SP} = .280, p < .001). Projects start as mini category and rapidly grow with their time in the market. The left part of figure 31 demonstrates a positive trend regarding growth, compared to market availability. The study shows that 37.09% of the projects have grown to a size larger than 10,000 LOC in their fourth year. Four.three two percent of the projects that are less than 0.5 years in the market belong already in the category of medium to large sized projects. A possible reason for this phenomenon could be that apart from high development work, these projects start as sub-projects or rapidly grow through the reuse of source code. The right part of figure 31 displays the percentile data grouped by each category. About 65.88% of the projects that are 2-4 years in the market have already grown larger than 10,000 LOC.

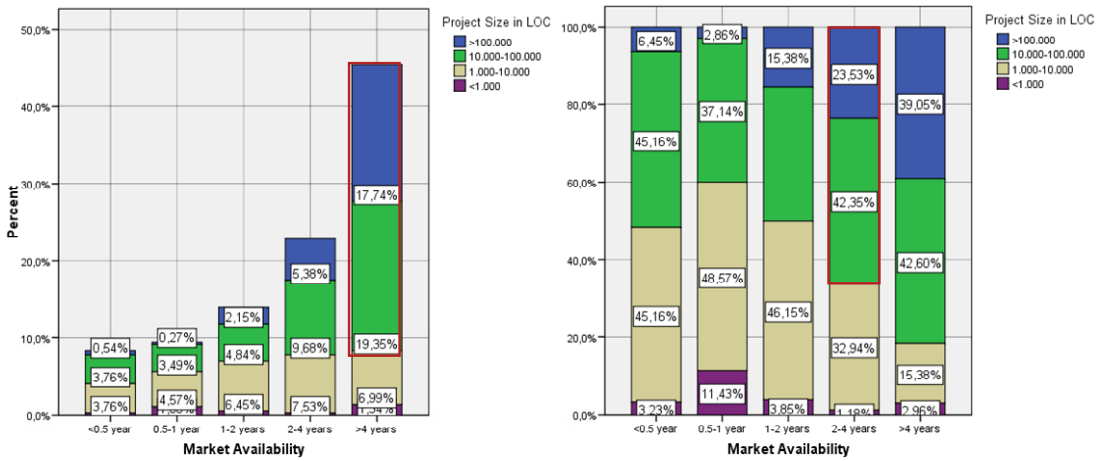

Figure 31. Project Size by Market Availability

The user community grows with the time of the project in the market (r_{SP} = .407, p < .001), similar to the project size. Roughly, 14% of the projects have more than 50 users in the first year, which constantly grows to 58.02% in their fourth year, as shown in figure 32.

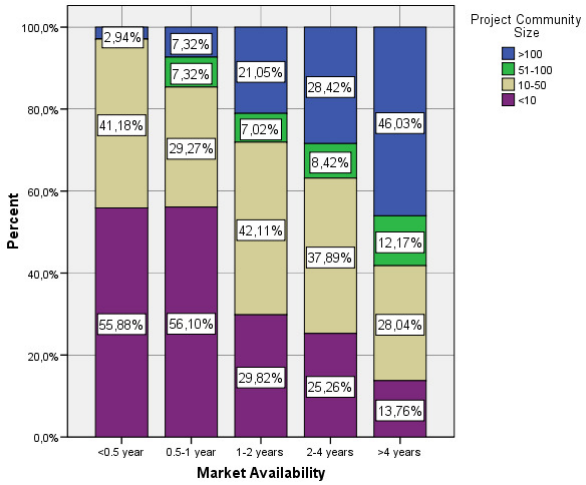

Figure 32. Community Size by Market Availability

The analysis of project maturity (G8) and market availability shows a positive correlation (c = .534, p < .001) and indicates how much time projects need to reach a productive status in the market. The productivity grows year by year from 14.7% in the category "<0.5 year" and reaches 57.8% after the first year. While some projects are still in the planning status after the first year, it is interesting to note that a small number of roughly 2% of each group is classified as inactive. The majority of projects reach a productive status after the first year, which is an impressive achievement for these OSS projects.

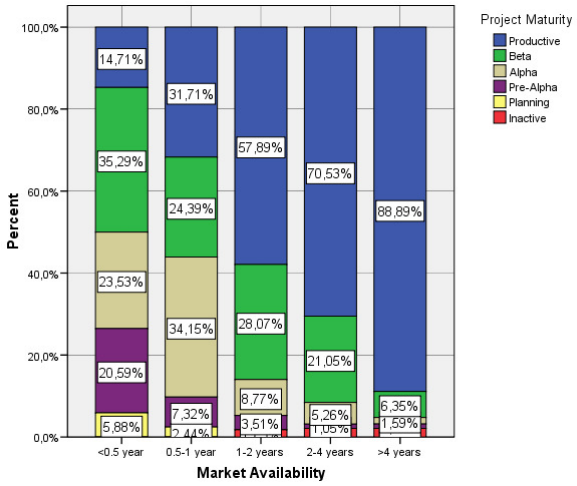

Figure 33. Project Maturity by Market Availability

6.2.2 Participants Classification

The type of participants who responded to this survey is an important perspective as it shows possible influences on the results and indicates the level of quality. This section explores the participant's role, the level of participation, the development experience and motivation to contribute to OSS projects.

The analysis of the respondents main role (S1) shows that 78.6% are a "developer" or a "project manager", 13.2% are responsible for "engineering" or "design" (as depicted in figure 34). Therefore, more than 91% of the respondents contribute to the development or management of the project, which corresponds to the aimed target group.

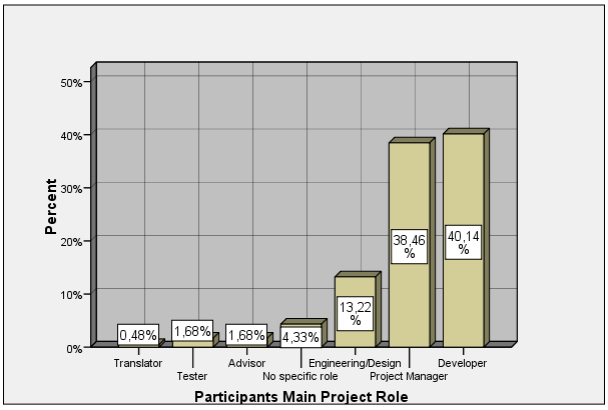

Figure 34. Participants Main Project Role

The majority of OSS projects are developed by 2-5 developers, which requires versatility from the participants. Small projects demand flexible adjustments to their needs and participants often hold different roles. A multiple question is used to capture all further roles (S2) each participant occupies. In table 8, additional roles are shown grouped by the participant's

main role. This question allows multiple answers. Therefore, each further role occurs to the maximum of the number of participants as displayed in their main role (N). For instance, two translators have marked that their further projects roles are “user”, “developer”, “contributor” and “tester”. In this case, the role “user” displays how many people with main role “translator” obtain this role.

Participants hold multiple roles, which provides evidence for versatile and fluid roles as argued by Jensen and Scacchi (2005). Furthermore, the table shows a significant proportion of “users”, which implies that many contributors are also users of their product.

Table 8. Participants Further Project Roles

Participant Main Role	Valid		Further Project Roles											
	N	Percent	No further role	User	Advisor	Engineering/Design	Developer	Core Developer	Contributor	Release Manager	Maintainer	Tester	Translator	Project Manager
Project Manager (S108)	153	100%	7	72	37	109	76	119	25	94	78	73	20	109
Engineering/Design (S104)	55	100%	0	24	16	42	34	44	4	32	22	25	2	27
Developer (S105)	158	100%	9	82	25	103	97	111	29	69	90	80	16	63
Tester (S106)	7	100%	1	5	0	1	3	0	3	0	0	5	2	0
Translator (S107)	2	100%	0	2	0	0	2	0	1	0	0	1	1	0
Advisor (S103)	7	100%	0	3	7	2	3	1	1	2	1	2	1	1
No specific role (S101)	18	100%	0	12	3	11	14	13	3	13	10	11	2	12

The level of development experience (S3) expresses the degree of knowledge and professionalism. Figure 35 depicts the level of development experiences, which shows that the majority of participants (75.2%) have more than five years of development experience, and provides consistency across the sample. This finding is an important quality and success criterion for the projects, as it is assumed that a high level of experience across the team may influence the approach and subsequently the defect density.

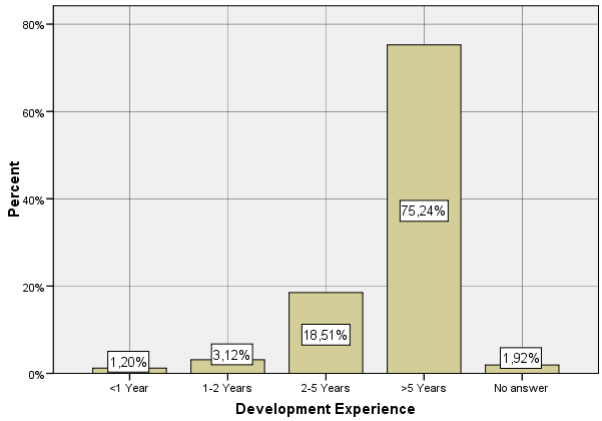

Figure 35. Participants Software Development Experience

The question is whether the development experience is influenced by the project size. The assumption is that large projects have a higher level of experience than smaller ones. A comparison of the development experience to the project size shows that no correlation exists (r_{SP} = -.040, p = .414). The level of experience is almost uniformly distributed, only mini projects have a higher proportion of less experienced developers than larger projects. However, these results could be negatively impacted through the small proportion of just 3.1% of small projects within the sample.

The analysis of the development experience in relation to market availability shows a weak correlation (r_{SP} = .208, p < .001). Figure 36 illustrates that the level of experience grows with the time in the market. This result is not unusual for software projects but could indicate the degree of fluctuation. The longer a product is in the market, the fewer will be the proportion of new, less experienced participants, while knowledgeable teams seem to accompany a project over a period. This may indicate a minimal fluctuation and a stable project organisation. Thus, the voluntariness of participation does not directly imply frequent organisational changes.

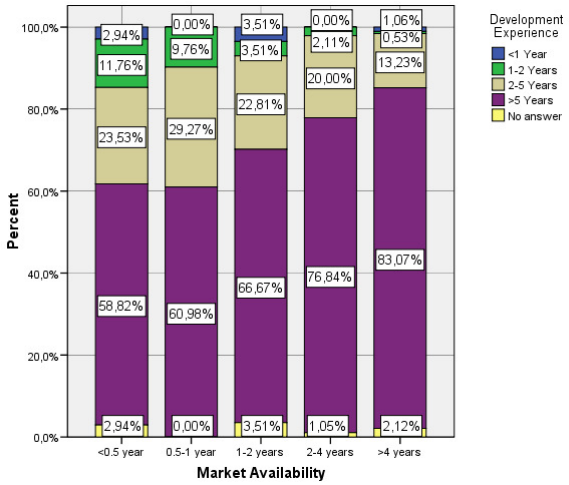

Figure 36. Development Experience by Market Availability

The examination of the project participation (S4) explains the contribution level of the participants. The study shows that around 54.8% work part-time on OSS projects, 20.4% contribute full-time to an engagement, while 24.8% work seldom or are passive contributors. A positive correlation (c = .302, p < .001) between the level of participation and the project size exists, which could provide evidence that companies interest rises with project growth. As depicted in figure 37, around 34% of the respondents contribute full-time to a large project.

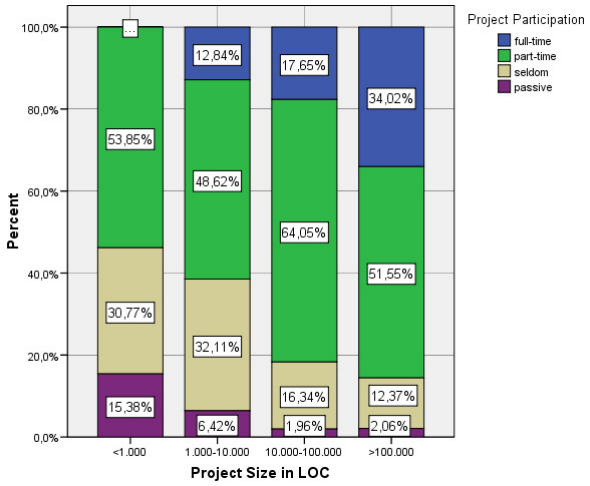

Figure 37. Level of Participation by Project Size

The analysis of the participant's motivation (S5) shows that 36.5% contribute for personal needs, 27.2% participate due to company needs and 22.1% report that they are motivated by community needs. A relatively large number of 14.8% of the participants state other reasons that could fit into the category personal needs, such as fun, education or research interests. Some state that all categories would apply to them.

A comparison of motivation to project size gives a weak positive correlation (c = .204, p = .115) reflecting that small and medium projects have a similar distribution when compared with the total result from all participants. It is noticeable, that a shift from personal to company needs appears with project growth, as depicted in figure 38. The result shows a corresponding picture to figure 37, which demonstrates an increasing participation with project growth. Commercial interests could be rewarded with the growth and maturity of a product, which explains the increase of company needs in large projects.

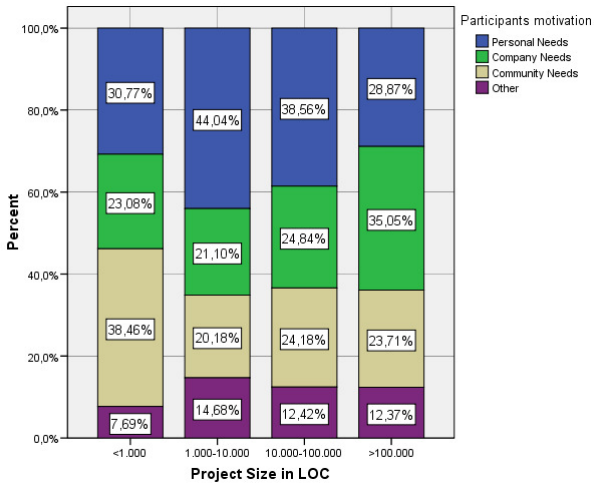

Figure 38. Participants Motivation grouped by Project Size

There is a weak positive correlation (c = .306, p < .001) of the participants motivation and their level of participation. Figure 39 illustrates that roughly 48% of full-time participants are contributing for company needs. This amount decreases with the level of contribution to 11.36% for "seldom" contributors, while the contribution for personal needs constantly grows with a decreasing level of participation from full-time to seldom.

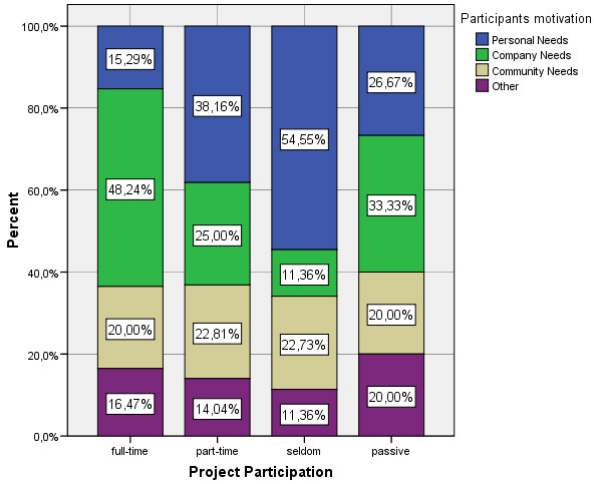

Figure 39. Level of Participation by Participants Motivation

The correlation between roles and motivation (c =.275, p = .012) proves how motivation differs between the roles. Within both large groups of "developer" and "project manager", the participant's motivations are uniformly distributed and reflect the overall result. The variety in the other groups might result from the sample itself. A high level of personal motivation could confirm that developers are also users of their products (see table 8). As displayed in figure 40, both "developers" and "project managers" contribute mainly for personal interests, which could provide evidence for this argument.

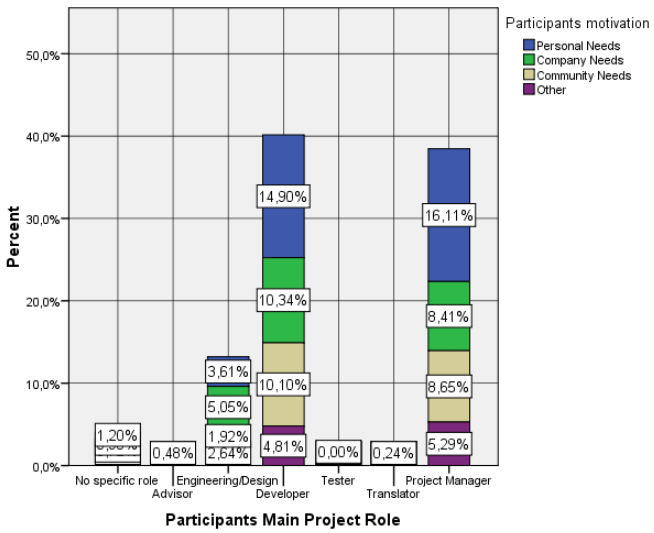

Figure 40. Participants Motivation grouped by Main Project Role

6.2.3 Processes

The OSS development model considers three main activity groups, such as planning, execution and release tasks (Ahmad and Lodhi 2006). Release activities differ between OSS projects due to the applied development language, as for instance PHP developments do not have a build phase. Therefore, the focus is on the design, development and testing activities (P1). A comparison of design (r_{SP} = -.079, p = .108), development (r_{SP} = .044, p = .374) and testing (r_{SP} = -.014, p = .776) activities to the project size shows that these groups do not differ significantly and are uniformly distributed. However, the proportion of the development activities increases with the project size, while testing efforts seems to decrease as shown in figure 41. This could indicate that large projects shift their activities to software development to the disadvantage of testing activities. A further analysis of the testing activities compared to the whole development time shows that testing efforts slightly decrease with project size from 40.3% in small projects to 38.5% in large projects.

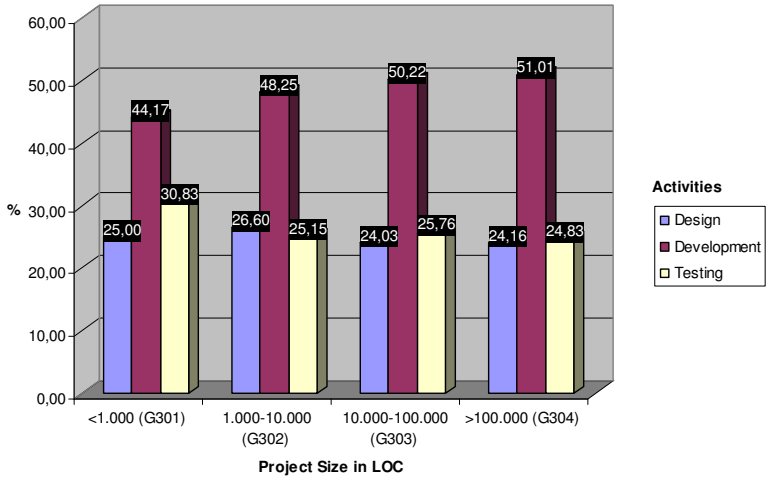

Figure 41. Proportions of Project Activities to Project Size

The examination of the release strategy (P4) shows, that 72.1% of the projects follow a feature-based strategy, which means that new releases are triggered with the readiness or maturity of the features, while a time-based approach (10.8%) follows scheduled release dates. Around 7% of the projects follow other strategies, mainly hybrid approaches, or combination of both. The results reflect the findings of the Working Group on Libre Software (2000) but illustrate that the proportions of a hybrid or time-based approach are not insignificant. There is a weak positive correlation of the release strategy and the project size (c = .251, p = .006), which indicates a trend of a time-based strategy in large projects (figure 42).

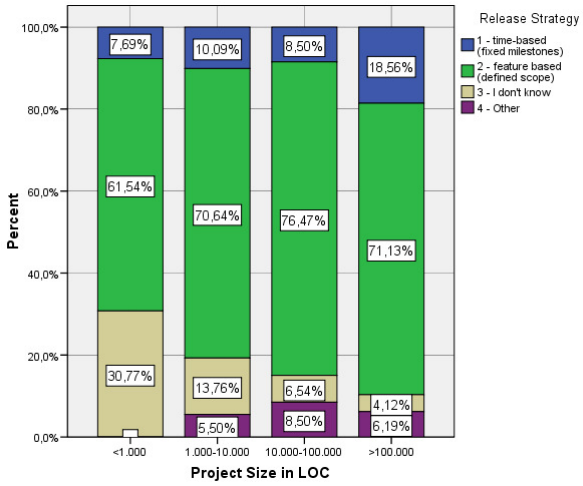

Figure 42. Release Strategy to Project Size

The observed release frequency (P2) confirms larger intervals as expected. Only 5.3% of the projects release once or twice every fortnight, 14.4% once a month, 29.6% once a quarter, the most significant number of 32.7% releases once half yearly and roughly 18% have even longer intervals. In contrast to Raymond's often-cited statement, "release early and release often" the observed release frequency is much lower, compared to the study of Zhao and Elbaum (2003), where 43% of the projects release every month.

There is a weak positive correlation of the release frequency with the project size which is not significant (c = .237, p = .208). Thus, the release frequencies are independent of the project size.

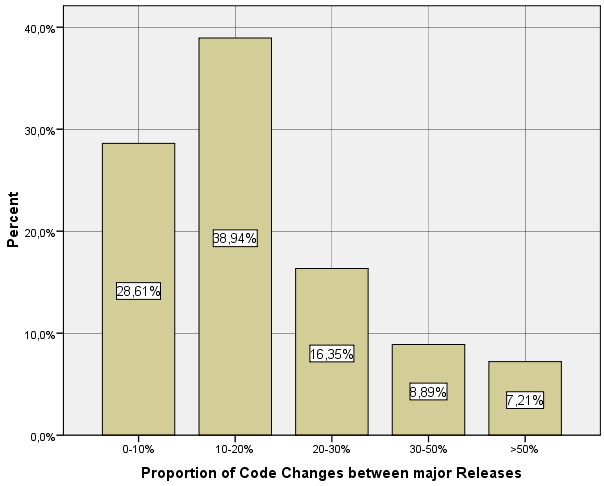

Figure 43. Proportion of Code Changes between major Releases

Figure 43 depicts the analysis of the code changes between major releases (P3) and shows that 28.6% of the total projects have minor code changes of 0-10%, 38.9% report 10-20% and 16.3% have code changes of 20-30%.

A comparison of release frequency to major code changes (r_{SP} = .068, p = .168) proves no correlation, that means major code changes are independent of the release frequency. A small number of projects releases every week or every fortnight, while most projects have longer intervals with changes up to 30%, as depicted in figure 44.

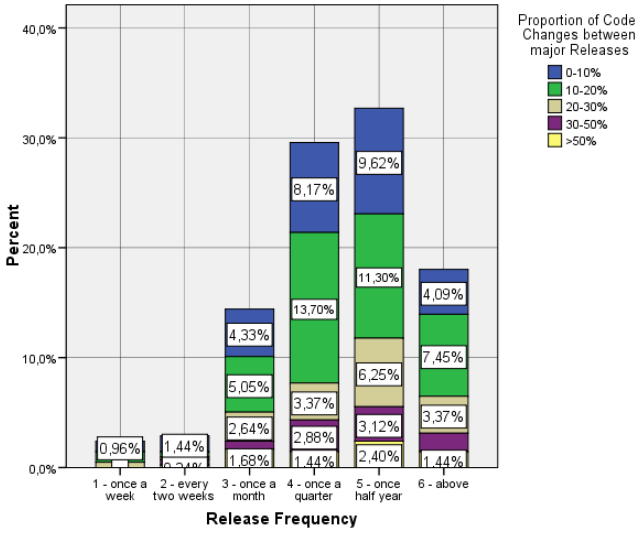

Figure 44. Release Frequency in relation to major Code Changes

The comparison of code changes in relation to the project size indicates the level of complexity that projects are facing. The average code changes vary from 10-20%. It is noticeable that in small to medium projects, more than 6% have major changes of 30-50% and more of their source code (figure 45). However, major code changes occur independently of the project size (r_{SP} = -.034, p = .487).

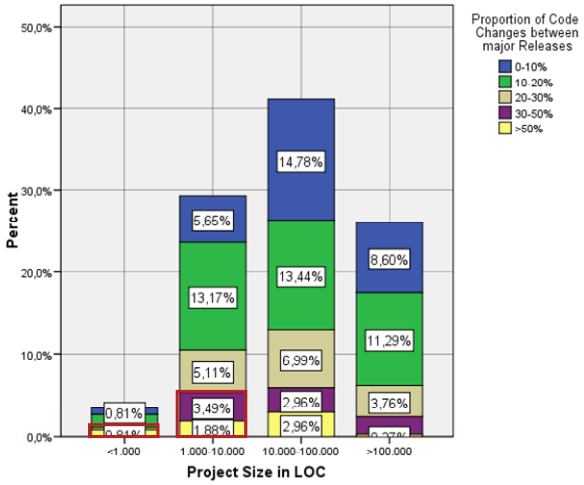

Figure 45. Code Change between Major Releases in relation to Project Size

Large code changes may result from intensive development or the reusability of source code (P5). The analysed OSS projects show uniformly distributed results (figure 46), which leads to the conclusion that code reusability is common and used by all projects but to different

extent. On average, the proportion of reused code is 31.7%. The reusability of code occurs independently of the project size (r_{SP} = .080, p = .106). A further analysis of the dependency of reused code in relation to testing effort is conducted in section 6.2.3.1.

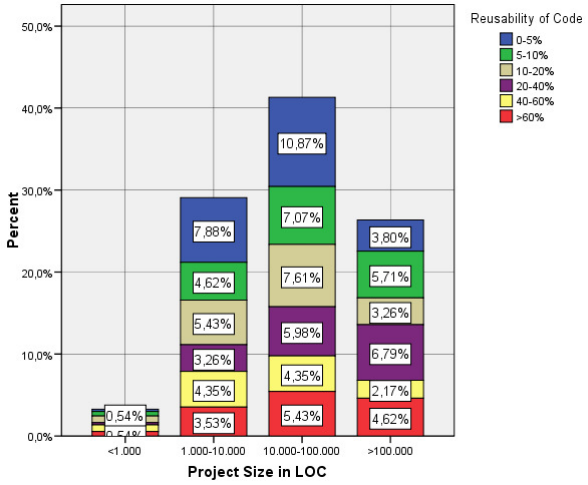

Figure 46. Reusability of Code in relation to Project Size

The modularity of the source code (P6) provides an indicator for the code quality. Hence, the point of time is important when modularity is considered as it significantly influences code quality. Figure 47 depicts the correlation of modularity to project size. The analysis shows a weak positive correlation (c = .212, p = .082). Thus, early consideration of modularity seems dependent on the project size. Roughly, 55% of the large projects define modularity already in the design phase, while 58.3% of the mini projects reconsider modularity during development. In the group of mini projects, 8.3% have no modular development. That leads to the assumption that mini projects starts from scratch and reconsider the development in later stages with product growth.

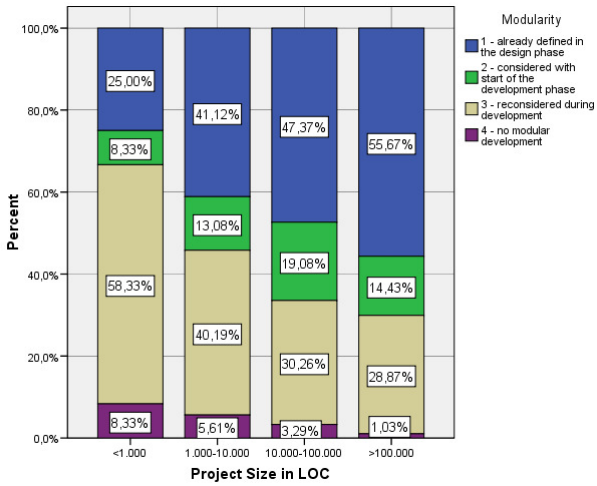

Figure 47. Modularity in Relation to Project Size

The examination of the proportion of abandoned code (P7) shows an average of 12.0%. It is assumed that with an increase of abandoned code, the code complexity increases (P8) as well, which could negatively affect the software quality. There is positive correlation between both variables (c = .378, p < .001). As displayed in figure 48, projects with an increased level of abandoned code report more issues. Hence, the level of abandoned code is an important quality indicator.

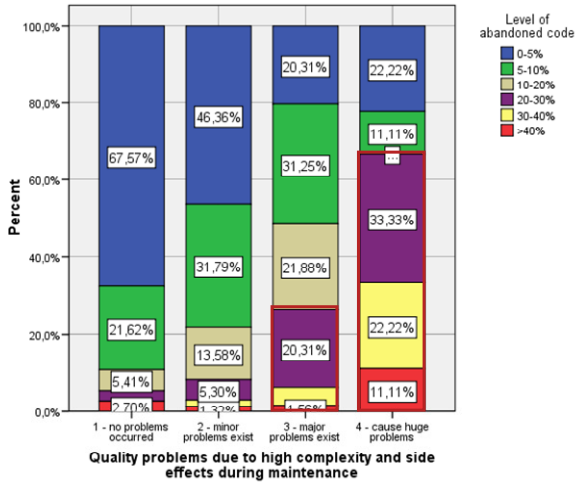

Figure 48. Code Complexity to Level of Abandoned Code

A further analysis of market availability in relation to abandoned code shows that the level of abandoned code does not significantly increase in accordance with the time of the products in the market (r_{SP} = .057, p = .246) but with the maturity of the product (c = .366, p < .001). Only projects that reach beta or productive status report the existence of abandoned code at all. Thus, issues with abandoned code are not time dependent but occur when projects reach a mature status.

6.2.3.1 Testing

This section focuses on testing approaches, such as walkthroughs, peer-reviews or inspections and examines the effect of user testing.

The average proportion of testing (T1) compared to development effort is 38.6% and corresponds to the findings of Zhang and Pham (2000) who observe that the majority of projects spend 20-40% in testing. However, these findings differ slightly from the result of Zhao and Elbaum (2003) who observe that larger projects spend less time in testing than smaller ones. Such tendency has been discussed in relation to project activities, as depicted in figure 41. However, as figure 49 illustrates, for this study the proportion of testing does not significantly correlate with the project size (r_{SP} = .002, p = .967).

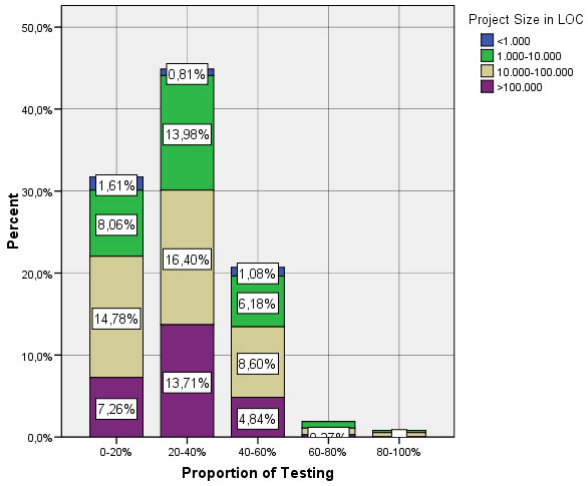

Figure 49. Proportion of Testing to Project Size

The assumption, that projects, which largely reuse code, could have decreased testing efforts, cannot be confirmed. The comparison of code reusability to testing effort shows a weak positive correlation (r_{SP} = .113, p = .021). Projects that largely reuse code seem to test larger proportions of their source code, as shown in figure 50. However, the scaled results in the category "testing proportion 80-100%" may be misleading due to less response in this category (compare with figure 49).

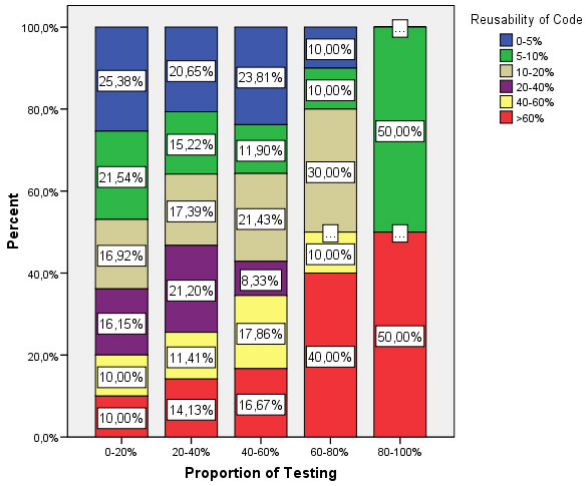

Figure 50. Proportion of Testing to Reusability of Code

More than 52.6% of the projects follow a structured testing (T2) approach, which comprises predefined test scripts or planning. Around 37.4% of the projects executed testing due to their experience, e.g. of previous projects and only 9.6% of the projects had no planning at all or did not know.

The average proportion of source code tested by the user (T3) is 55.7%. However the statistical correlation between user testing and project size is weak and less significant (r_{SP} = .086, p = .080). Figure 51 illustrates that large projects seem to benefit more from user testing, as

roughly 40% report that more than 60% of their source code is tested by users in contrast to small projects with a proportion of 33.9%. Mini projects with less than 1,000 lines of code have been excluded due to less response within this category. Further analysis is required to prove this assumption and to explore the relation of project size to testing.

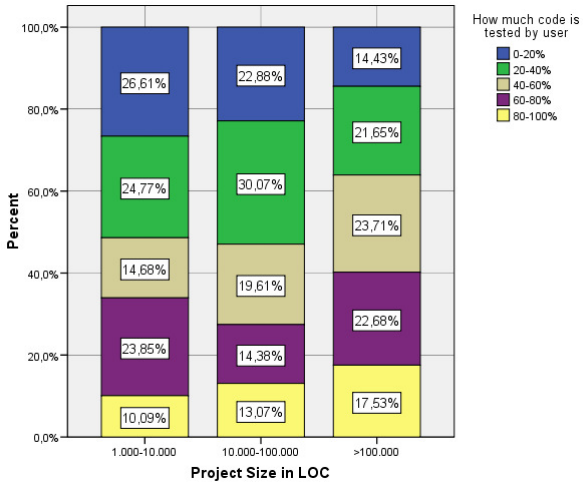

Figure 51. Code tested by Users compared to Project Size

User suggestions (T4) are a significant element in testing. Forty eight and a half percent of respondents say that user suggestions "bring the design forward", but 49.2% feel that "user suggestions are only sometimes useful". Only 2.2 % answered that user suggestions are not very efficient nor do not fit into the design. The result proves that developers follow Raymond's rules and clearly "listen to their customers" (Raymond 2001).

The efficiency of the user testing (T5) is regarded from the developer perspective. Only 4.6% of the respondents state that the users could not help much, 32.7% mention that user testing supports them but they would have found the bugs on their own. However, 57% of the participants report that the users found major defects or hard bugs, which confirms a high user testing efficiency. This finding needs to be set against the degree of tested code by the users. The assumption is made that the more users observe code the more defects are found. The analysis displays a weak positive correlation (c = .258, p = .004). Figure 52 shows that more code is tested in the group of projects that reports their "users find hard bugs". This finding supports the often-cited statement attributed to Torvalds that "Given enough eyeballs, all bugs are shallow" and shows the high efficiency of user testing.

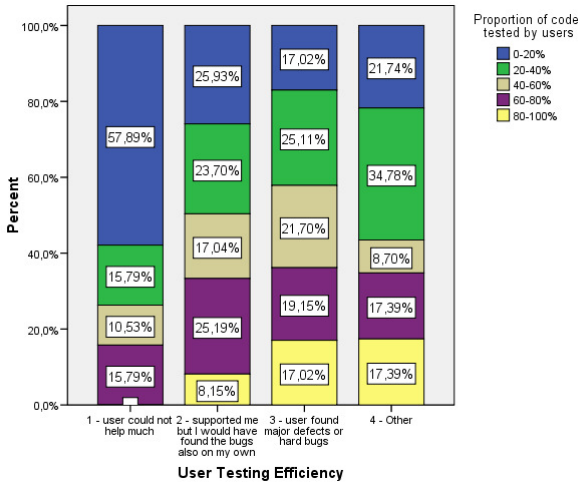

Figure 52. User Testing Efficiency to Proportion of User Testing

Projects seem to benefit more from user testing than from developer reviews. The comparison of code tested by user to code reviewed by developers shows that on average 55.7% of the code is tested by users, while developers review only 47.2% of the code (see table 9).

Table 9. Code Tested by Users and Developers

Category			Frequency	Percent	Valid-Percent	Proportion of code tested	Average Proportion in Percent
Code tested by users	Valid	0-20%	92	22.1	22.1	18.4	4.42
		20-40%	102	24.5	24.5	40.8	9.81
		40-60%	80	19.2	19.2	48	11.54
		60-80%	86	20.7	20.7	68.8	16.54
		80-100%	56	13.5	13.5	56	13.46
		Total	416	100.0	100.0		**55.77**
Code reviewed by developers	Valid	0-20%	169	40.6	41.0	33.8	8.20
		20-40%	91	21.9	22.1	36.4	8.83
		40-60%	45	10.8	10.9	27	6.55
		60-80%	47	11.3	11.4	37.6	9.12
		80-100%	60	14.4	14.6	60	14.56
		Total	412	99.0	100.0		**47.26**
	Missing	System	4	1.0			
	Total		416	100.0			

Code reviews (T6) are conducted by 69.9% of the projects. Just over twenty-one percent of the projects apply frequent reviews, additionally 25% inspect only important or critical code and 23.7% perform reviews seldom. It is obvious that projects with a higher inspection rate of their source code apply code reviews more frequently. There is a significant positive correlation of inspected code (T7) to the application of reviews ($c = .633$, $p < .001$). Figure 53 demonstrates that projects that use code reviews more frequently inspect more than 60% of their code.

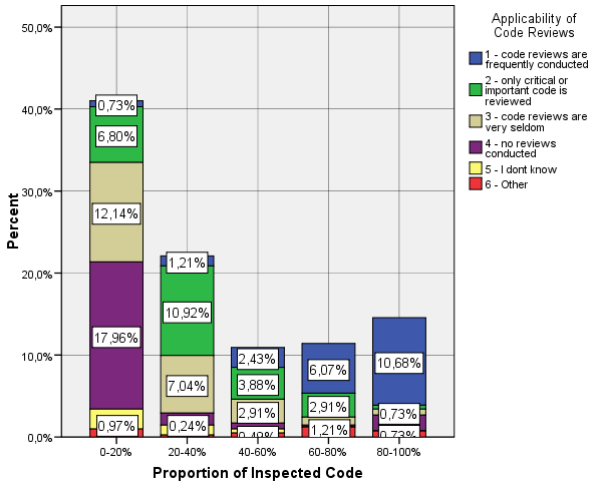

Figure 53. Proportion of Inspected Code to Applicability of Reviews

The peer review technique (T8) is used by 50.5% of the respondents. Around 14.8% claim to use peer reviews as part of their testing approach, 17.9% conduct them on request and a further 17.7% apply peer reviews only for critical source code. This finding does not correspond to Raymond's (2001) observation, which claims a high degree of user involvement and peer reviews in OSS projects. A further analysis is conducted to examine this finding in relation to project success (refer to chapter 6.2.6).

In comparison to peer reviews, code reading or walkthroughs (T9) are more frequently applied (61.4%). Around 11.8% say that code readings or walkthroughs are their standard procedure, 27.4% use them sometimes and 22.9% use them in special cases.

The analysis of additional review processes, such as code control before commit (T10) shows that roughly 46% of the projects perform corrective actions before code commit. This means 12.3% reject inappropriate styled or structured code and 33.7% rework the code by their lead developer. Other projects (13.3%) integrate the code with annotations but 26.7% accept the code without comments. Projects approach code control processes differently, which depends not least on their chosen access restrictions to the code repository. Such differences may refer to organisational size, as a weak positive correlation exists (c = .277, p = .025). Figure 54 shows that 75% of the large projects execute corrective actions before code commit.

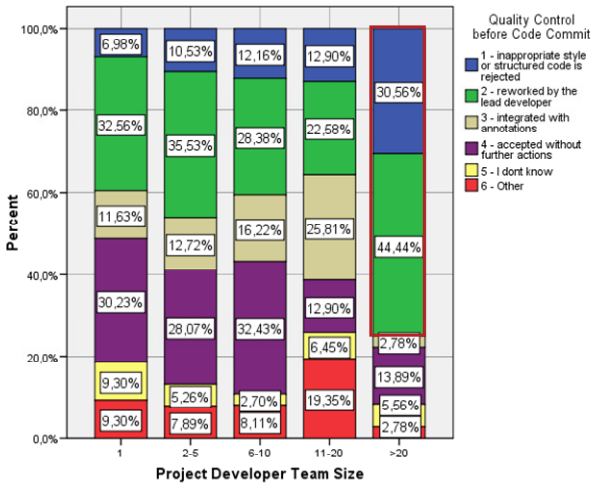

Figure 54. Quality Control before Code Commit to Developer Team Size

6.2.3.2 Defect Handling

Within this section the defect handling processes, such as bug fixing and tracking activities with focus on scope, process, approach and their efficiency are covered.

Effective defect handling, which leads to acceptable response time and reliable products, is one beneficial aspect from the development potential of the community and contributes to product quality. The timely introduction of defect handling (T11) provides insights about the organisational approach. Only 31.7% of the projects introduce defect-handling processes directly at the start of the project, while 64.5% introduce these activities during the project as displayed in table 10.

Table 10. Introduction of Defect Handling

	Category	Frequency	Percent	Valid Percent	Accumulated Percent
Valid	1 - directly from project start	132	31.7	31.7	31.7
	2 - with start of the design phase	25	6.0	6.0	37.7
	3 - with start of development/coding phase	135	32.5	32.5	70.2
	4 - with start of testing phase	65	15.6	15.6	85.8
	5 - in the post-release phase	43	10.3	10.3	96.2
	6 – Other	16	3.8	3.8	100.0
	Total	416	100.0	100.0	

Figure 55 shows that the main source code issues are within the scope of defect handling (T12). Nearly eighty nine percent of the projects track code issues, while other issues such as requirements (41.8%), designs (43.0%) or documentation issues (48.5%) are tracked on average by 44.4% of the projects.

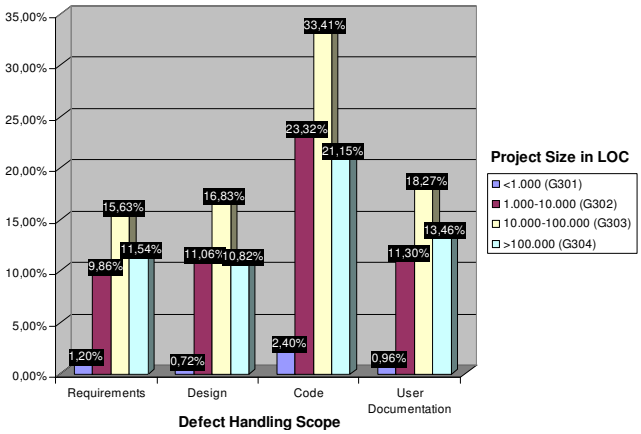

Figure 55. Defect Handling Scope

The examination of the defect reporting quality (T13) indicates that more than 51% of the projects report having useful defect descriptions with additional environmental information. Additionally 34.9% obtain at least short bug descriptions from their testers, while 13.8% suffer from unstructured or not useful information, as illustrated in table 11.

Table 11. Defect Handling Scope

	Category	Frequency	Percent	Valid Percent	Accumulated Percent
Valid	1 - often detailed information with priority and majority	59	14.2	14.3	14.3
	2 - often useful bug descriptions with environmental information	151	36.3	36.7	51.0
	3 - often only short bug description	145	34.9	35.2	86.2
	4 - often unstructured information	49	11.8	11.9	98.1
	5 - often no useful information at all	8	1.9	1.9	100.0
	Total	412	99.0	100.0	
	Missing	4	1.0		
	Total	416	100.0		

The examination of the defect handling time (T15) in relation to the defect reporting quality indicates whether the reporting quality correlates with the handling time. The correlation is not significant (c = .186, p = .252) but shows tendencies that the response time decreases with decreasing quality as depicted in figure 56. Correspondingly, the proportion of immediately solved defects shrinks to zero when information quality is poor.

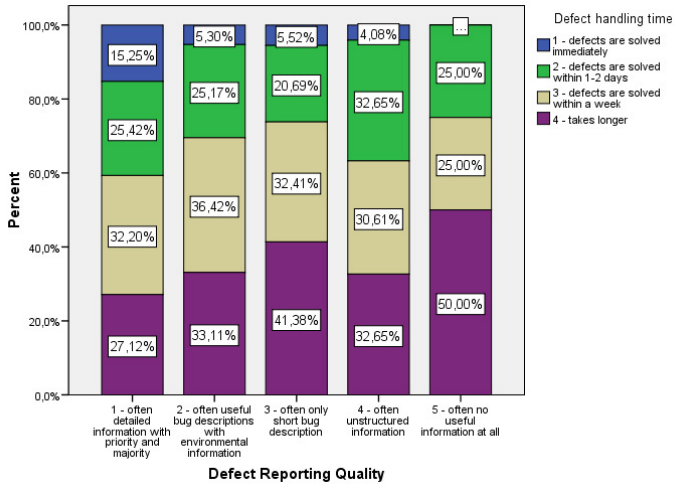

Figure 56. Reporting Quality in Relation to Defect Handling Time

The management and organisation of defect handling processes (T14) differs within the observed projects. A small group of projects (11.5%) cluster their topics and prioritise them before they are assigned to a developer. The majority of projects (50.9%) follow a simple assignment of issues to the developers, while 27.4% say that everyone is responsible, which falls into the same category of projects (10.0%), who report having no defect handling management. The tracking of the defect status (T17) seems common practice, as 57.4% reply they have rules for categorization and status follow-up. Another 19.7% purport to classify at least some of their issues, while roughly a quarter (22.8%) have no classification of defects at all.

The handling of security critical defects (T16) (when appropriate in the projects) shows that 10.0% of the respondents "can hardly determine the criticality of these issues", while 8.9% follow a structured approach and have "clear escalation procedures with assigned responsibilities". Some other respondents replied that security defects are prioritised against business needs. Figure 57 shows that more than 44% solve these issues across the whole development team or have no special treatments (28.8%).

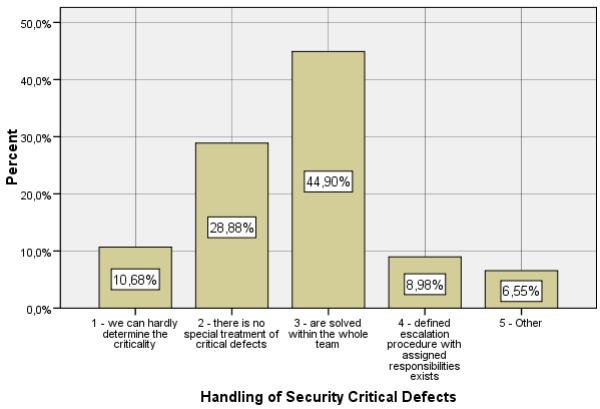

Figure 57. Handling of Security Critical Defects

6.2.4 Organisation and Infrastructure

In this section, organisational and infrastructure topics, such as internal project communication, integration of new participants, documentation, infrastructure and tools are discussed.

6.2.4.1 Knowledge Transfer and Communication

The term "on-boarding" describes the process of taking new participants on to the project and comprises all activities for effective knowledge transfer. Thus, new developers obtain all necessary information to work efficiently, directly from the start. The analysis shows that only 27.4% report having on-boarding procedures (C1), 27.8% are unsure and 44.7% had no such processes. The correlation of on-boarding processes to project size is not significant (c = .161, p = .196). Figure 58 indicates a higher availability of on-boarding processes with project growth. This may result, either from the necessity to counteract the lack of knowledge or simply to reduce on-boarding time. For large projects, 63.92% of the respondents state that they do not have any procedures (or could not tell) which is surprisingly high and may indicate that OSS projects do not suffer from knowledge transfer problems as expected. This could provide evidence that uncertainties regarding developer fluctuation are less acute.

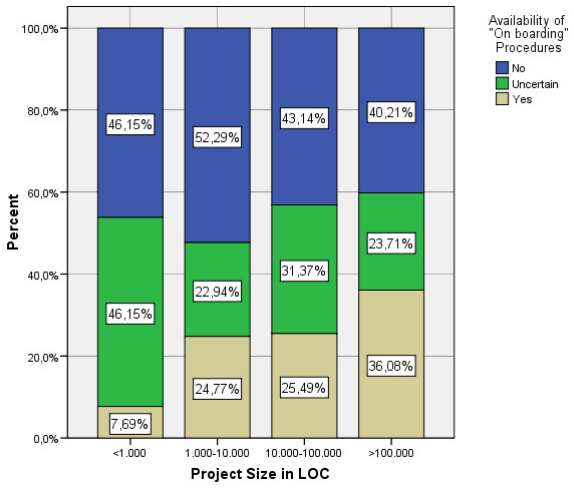

Figure 58. Availability of "On-boarding" Procedures by Project Size

The "on-boarding time" (C2) describes the period until new participants reach full productivity. Figure 59 indicates that the time increases with growth of the project in size and knowledge, as expected (r_{SP} = .156, p = .001). Mini projects require roughly up to one week, while large projects face periods of 2-4 weeks to take new participants on to the project.

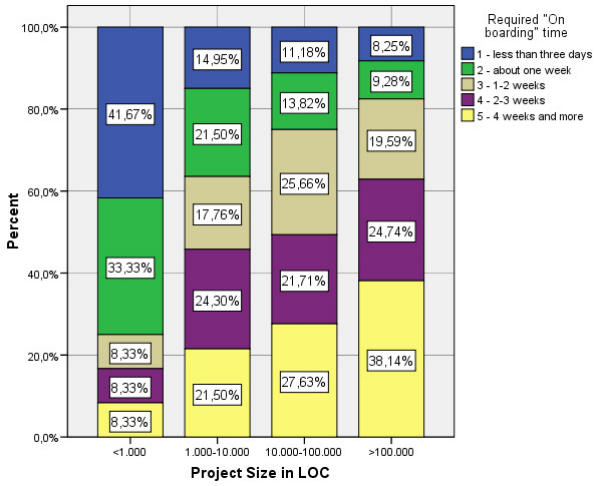

Figure 59. "On-boarding" Time by Project Size

It is assumed that there is a relation of the existence of "on-boarding" procedures to the "on-boarding" time. However, the weak positive correlation is not significant (c = .147, p = .329), which means that the on-boarding time is not affected by the existence of such processes but rather through the project size, as shown in figure 60.

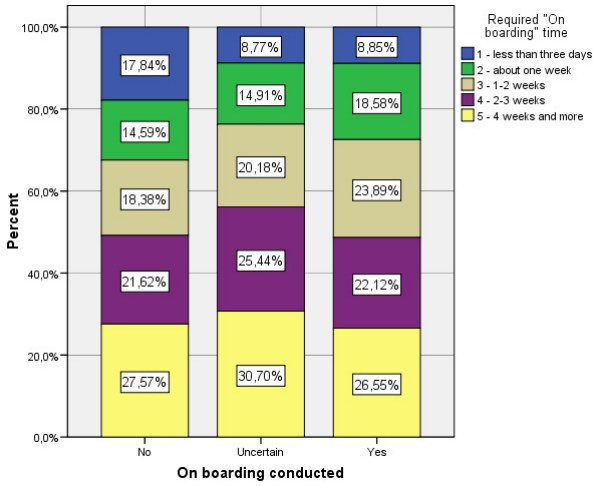

Figure 60. "On-boarding" Time by "On-boarding" Procedure

Beside an effective knowledge transfer to developers, it is of interest how the respondents rate the communication between developers and users. Following Raymond's (2001) statement "listen to your customers", efficient communication (C3) is regarded as an important element to retrieve information about requirements or development issues from the users. In total 72.6% of the participants report a direct and efficient feedback between developers and users, while 19.7% are uncertain and only 6.7% deny this. The comparison of communication efficiency in relation to project size in figure 61 shows a surprising result, as the perceived communication efficiency seems to increase with project size, which contradicts expectations (c = .215, p = .213).

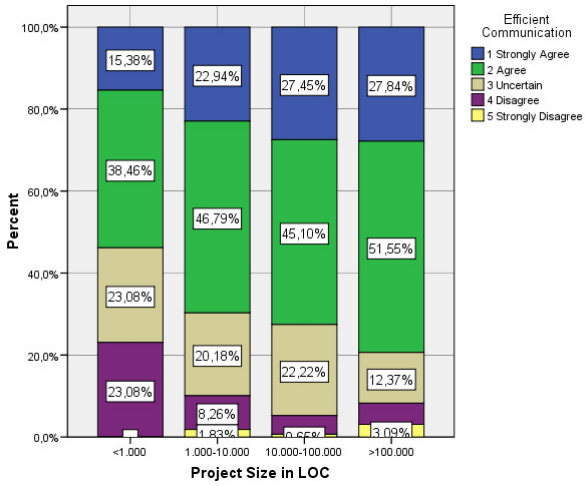

Figure 61. Efficient Communication in Relation to Project Size

6.2.4.2 Documentation

The examination of documentation focuses on processes, development guidelines and on the product. Around 58% of the projects have, at minimum, minor process descriptions, while 41.3% claim to work mostly without process descriptions or even argue that "processes are common and do not need to be documented". Figure 62 illustrates the results of the availability of process documentation (D1).

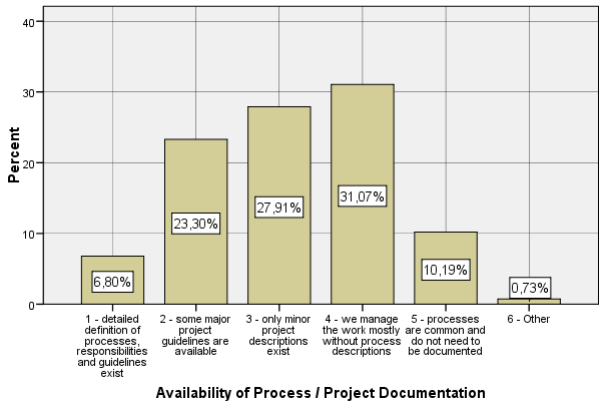

Figure 62. Availability of Process Documentation

Defined processes are more frequent in larger projects, as shown by the weak positive correlation of process documentation to project size (c = .288, p = .011). More than 71% of large projects have a focus on process description, as depicted in figure 63. In contrast, 65% of the mini projects manage their projects without any process description. These findings clearly indicate a growing existence of process documentation with increasing project size. Common process descriptions could help large communities to counteract their increased complexity, through standardisation, knowledge sharing or to ease on-boarding procedures.

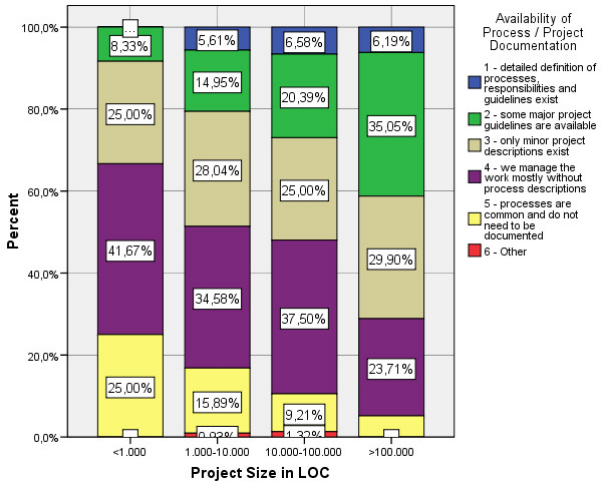

Figure 63. Process Documentation in relation to Project Size

OSS projects seem to have a strong focus on development style or coding guidelines (D2) as 65.3% of the respondent's report (see table 12).

Table 12. Development Documentation

		Frequency	Percent	Valid Percent	Accumulated Percent
Valid	1 - coding styles and development guidelines widely available	104	25.0	25.2	25.2
	2 - minor guidelines exist	165	39.7	40.0	65.3
	3 - no guidelines needed as developers are experienced enough	117	28.1	28.4	93.7
	4 - I don't know	16	3.8	3.9	97.6
	5 - Other	10	2.4	2.4	100.0
	Total	412	99.0	100.0	
Missing	System	4	1.0		
Total		416	100.0		

A similar result can be observed when comparing the availability of development documentation to the project size (c = .309, p < .001). Mini projects almost seem to neglect development guidelines (83.2%), while 83.5% of the large projects have development or coding style guides, as shown in figure 64. These findings correspond to Michlmayr's work (2005), which observes appropriate process documentation in large projects.

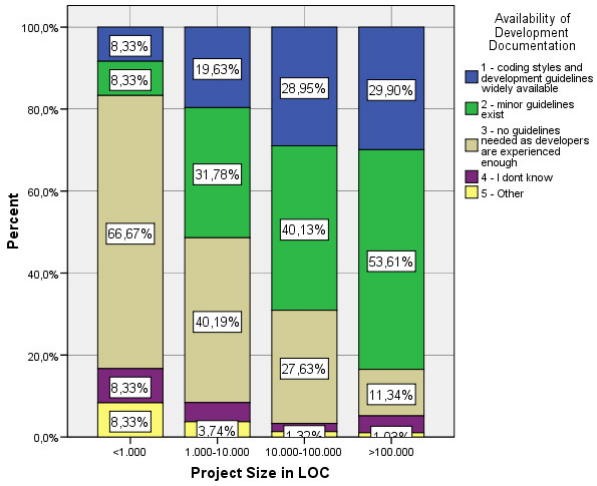

Figure 64. Development Guidelines in Relation to Project Size

The term "product description" comprises all relevant documentation for the usage of the product, such as user documentation, guidelines or technical documentation. The availability of an adequate product description may influence the acceptance of the product in the market. Hence, inadequate descriptions may limit less technically oriented users from using the software hence restricting its usage. Therefore, this criterion is regarded as an important quality characteristic. In total, 63.6% respond to have detailed product documentation (D3) or at least documented important parts. The weak positive correlation (c = .249, p = .036) shows that product documentation growth with the time in the market, as illustrated in figure 65. However, 25% of the surveyed projects report having only incomplete drafts and 10% are in this situation even after four years in the market.

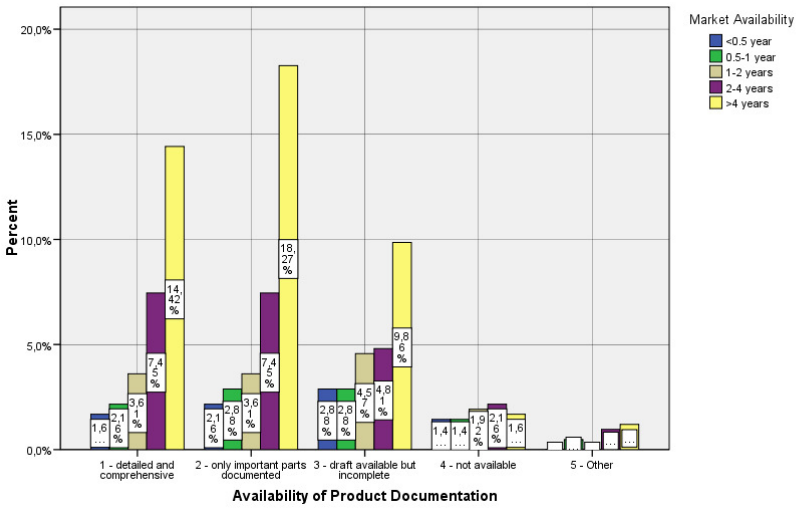

Figure 65. Product Documentation by Market Availability

6.2.4.3 Tools

The usage of tools (I1) plays an important role in the OSSD lifecycle and has important quality implications for processes as well as for the organisation, such as the effective support of communication or cooperation. The survey explores eight categories of collaboration tools with focus on source code, testing, infrastructure and communication.

In general, the observed tool usage is relatively high, compared to the findings of Zhao and Elbaum (2003). In total, 87.2% of the projects apply Source Code Control Tools, followed by Bug-tracking tools (76.2%) and Mailing Lists (73.5%), as depicted in figure 66.

Source Code Control Tools, such as Subversion (56.7%) or CVS (34.9%) clearly dominate the market. Projects mainly use SourceForge facilities (46.1%), followed by Bugzilla (16.8%), Trac (8.6%) or Jira (7.3%) as "Bug Tracking Tools". The most popular Mailing List tool is SourceForge Mailman (76.3%), while the rest of the projects use custom made or other solutions.

Web Portals play an important role for 54% of the projects, which is surprisingly low, because the predefined target group comprises a large proportion of projects hosted on SourceForge. Popular Web Portals are SourceForge (47.7%), "Wiki" products (15.4%), Trac (5.0%), while the rest use custom made or other web portals.

Instant Messaging Tools, such as Internet Relay Chat (IRC), MSN Messenger, Skype, Google Talk, Gaim, Jabber or ICQ are used by 51.9% of the projects. The well-known products in the group are IRC (21.6%), MSN (18.3%) or Skype 11.3% followed by other established products.

Only one third of the projects use Test Support tools, Code Viewers or Automatic Build Tools. A multiplicity of "Automatic Build Tools" is used by OSS projects, such as Ant (21.3%), Make (9.1%), Maven (7.6%) or CruiseControl (6.6%).

The analysis of the proportion of Code Viewers compared to Source Code Control Tools questions whether the respondents interpret the question equally, because renowned "Source Code Control Tools" offer basic code viewers. The purpose of the question is to evaluate whether and what kind of code viewers are applied. The results show, that ViewVC (17.3%), Subversion (13.6%), Trac (8.7%) or Eclipse (8.0%) are the most dominant products in this area.

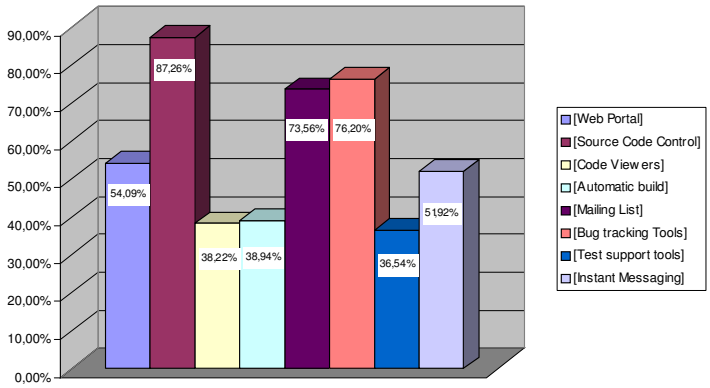

Figure 66. Tool Usage

The comparison of tool usage in relation to project size shows that large projects make more use of supporting tools in the development lifecycle, than smaller ones. Further supporting graphics can be found in the Appendix (A.4). Source Code Control Tools, as well as bug tracking tools, seem to play an important role with the growth and increasing complexity of development projects (c = .282, p < .001), as shown in figure 67.

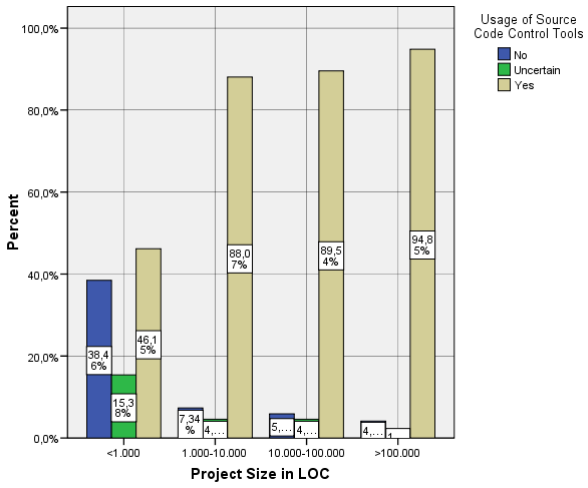

Figure 67. Usage of Source Code Control Tools in relation to Project Size

Code Viewers are more commonly applied in projects that perform frequent code reviews or inspections. The weak positive correlation shows (c = .252, p = .002) that review processes are supported by code viewers.

The usage of bug tracking tools correlates with project size (c = .247, p = .001). Bug tracking tools are more frequent in larger projects, which corresponds with the findings of Zhao and Elbaum (2003). The majority of mini projects renounce the applicability of Bug Tracking Tools, which may result from a manageable level of code complexity.

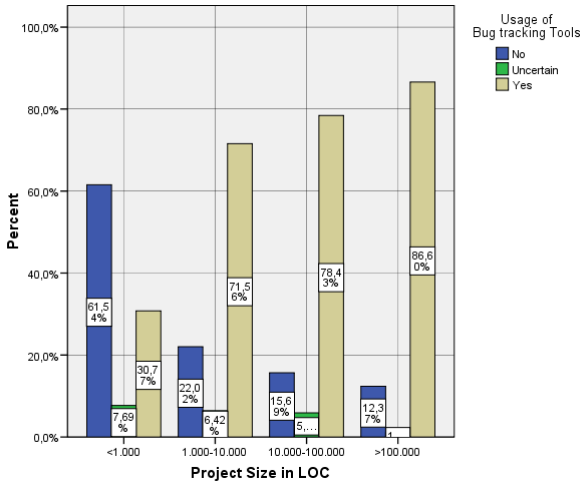

Figure 68. Usage of Bug Tracking Tools in relation to Project Size

The applicability of Bug Tracking Tools positively correlates with the defect handling process (c = .294, p < .001), the defect reporting quality (c = .263, p < .001) and the defect handling time (c = .221, p = .002). Projects that apply Bug Tracking Tools on average report receiving more structured information and have a better reporting quality, as shown by the fact that 54.3% claim to have classified and harmonised defect information (figure 69).

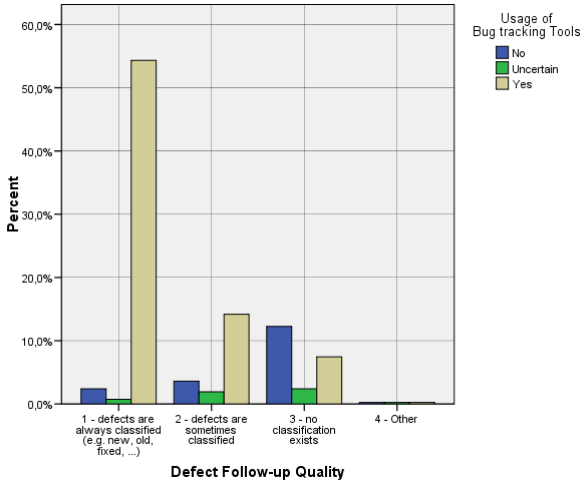

Figure 69. Usage of Bug Tracking Tools for Defect follow-up

Figure 70 shows the relation of defect reporting quality to bug tracking tools. Projects that implemented bug-tracking tools report a higher defect reporting quality.

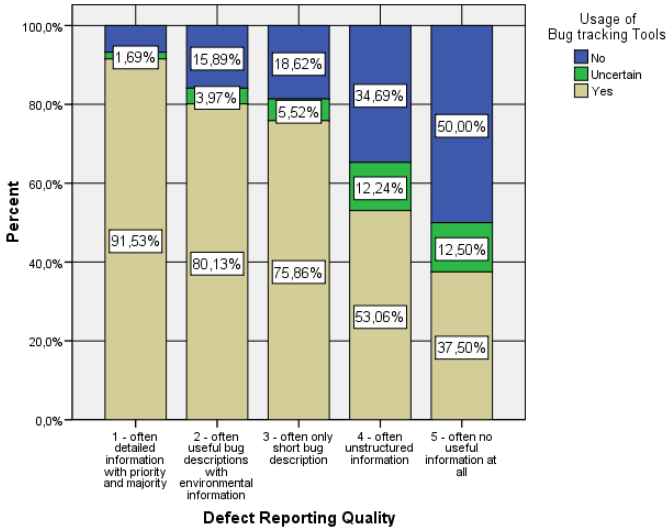

Figure 70. Usage of Bug Tracking Tools to Defect Reporting Quality

Projects that report having a structured testing approach use largely test support tools (c = .453, p < .001) as depicted in figure 71. Correspondingly, projects without supporting tools mostly executed testing according to their experience.

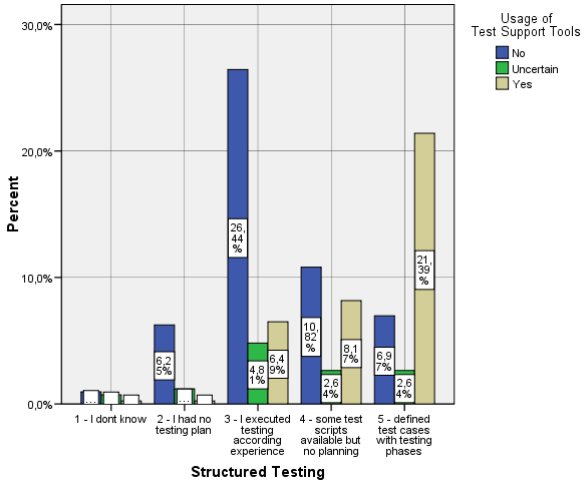

Figure 71. Structured Testing in Relation to Test Support Tools

In total, only 36.5% apply Test Support Tools, which does not play a dominant role compared to other tools. One reason could be that no standard has been established and beside JUnit (28.8%) a multiplicity of tools is employed. However, the usage of testing tools correlates to the project size (c = .227, p = .004). Mini, small or medium sized projects mostly negate the usage, while the trend shows clearly the growing applicability with increasing project size (figure 72).

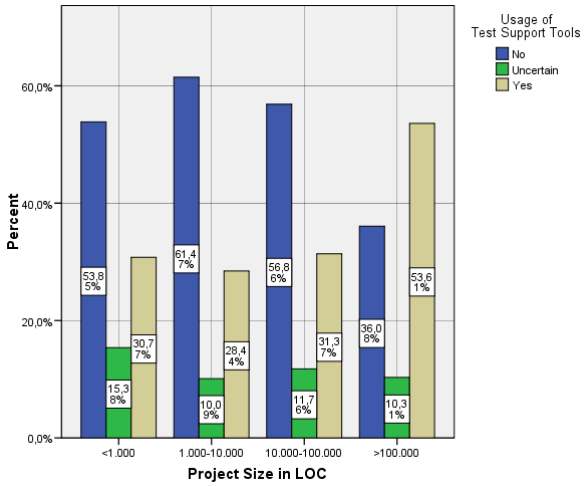

Figure 72. Usage of Test Support Tools in relation to Project Size

6.2.5 Quality Assurance

This section discusses the Quality Assurance topics with the focus on the applied practices, adherence to standards and suggested improvements.

Quality Assurance practices (Q1) are executed by 31.5% of the surveyed projects. Most of the projects do not have QA issues in their focus and the comparison of the variable "QA processes" in relation to the project size shows no significant correlation (c = .054, p = .598). Figure 73 indicates a clear trend with project growth, which could not be statistically proven.

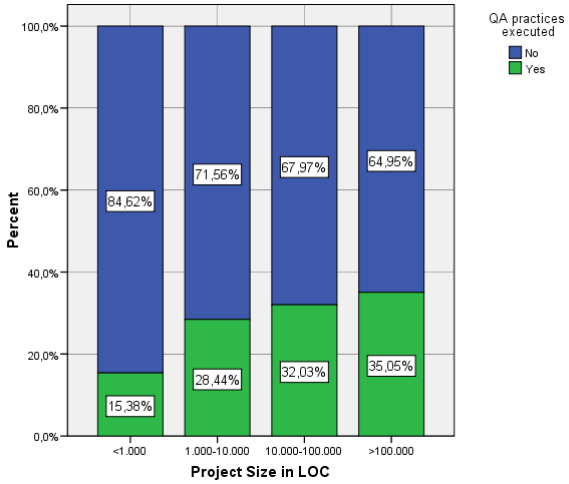

Figure 73. Quality Assurance Practices by Project Size

The analysis of the question "whether the Quality Assurance team checks that the product adheres to standards and requirements" (Q2) shows a similar result. QA checks are performed by 33.4% of the projects but do not significantly correlate to project size (c = .107, p = .302).

An indicator for the efficiency of the QA team may indicate the improvement activities (Q3), which affect the team or processes. One-fifth (20.2%) of the respondents reply that improvement actions are performed by their QA team. This does not significantly correlate to the project size (c = .146, p = .060). Compared to the group of projects that apply QA practices, 86.9% of them actively trigger improvements (c = .515, p < .001). This shows that projects which perform QA practices benefit from further improvements.

In a closed-ended question with an ordered scale, the respondents were asked to evaluate the applicability of each attribute or statement, according to their experience. The results illustrate the average rating of each statement, as depicted in figure 74. The responses show a predominant agreement with the statements, which might be positively influenced from the bias in the question due to loading in one direction. Hence, these findings are not used for further statistical analysis, but are displayed to give a first impression of the participant's position.

Well-defined processes as well as the applicability of Source Code Control Tools, Testing Tools or Mailings Lists are highly rated. The respondents rate user testing very highly. However, user defect reporting quality scores less. The bias of the peer review question might indicate that the respondents judge peer reviews to be less effective than normal reviews. Similar conclusions may result from the evaluation of automatic build tools, code viewers or web portals. Most of the respondents agree that frequent release cycles have a positive influence on software quality. However, this view ignores the conflict between quick user responses and short development time.

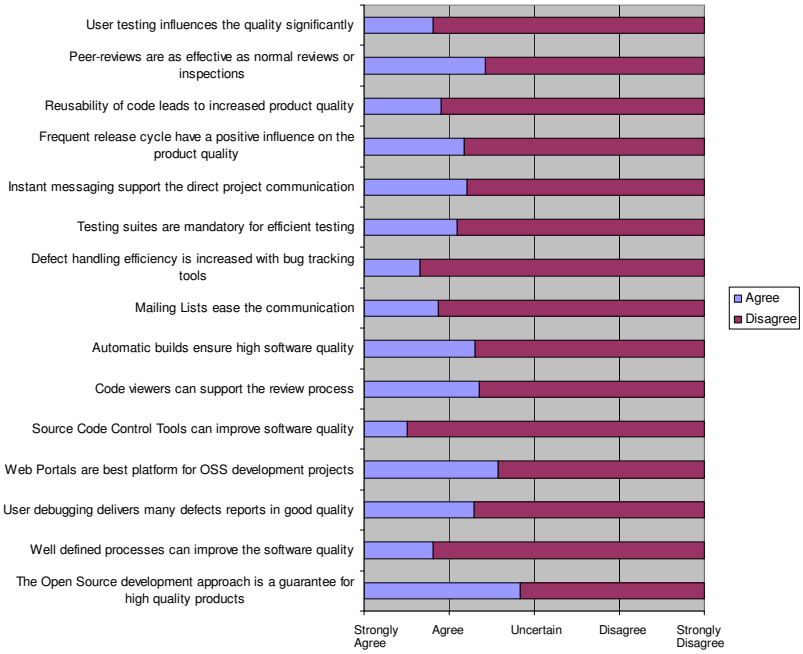

Figure 74. Participants Quality Evaluation

6.2.5.1 Quality Practices

Open-ended questions were used to examine the respondent's experiences about their applied quality practices, their performed quality assurance actions as well as their suggestions for quality improvements.

Within this section, the applied quality practices are discussed, clustered by community, development, documentation, release management, testing, tools and QA aspects. Several respondents replied that quality relevant issues are discussed within the core development team or the whole community. Some projects focus on the enhancement of their community, recruiting new volunteers for further test support. The definition and adherence of processes and standards, as well as changes of the development approach to e.g. Rational Unified Process (RUP) or a Test-Driven Development (TDD), are mentioned. Controlling tools are implemented to improve the defect handling. Some projects establish professional version management or conduct code freezes followed by intensive testing, to improve their software quality. Other projects implement change management processes, to keep control of the source code and the requirements.

Several projects review the structure of their project organisation or define new roles, such as a "QA team member", who handles pre-release tests and supports the release manager. Other projects increase the capacities of their QA members towards a full-time contribution, to perform reviews and checks to an appropriate extent. The setup of QA reviews and checks, to ensure the adherence to processes and standards, as well as a continuous improvement process are triggered.

Several projects conduct frequent code reviews of specific parts, previous fixes and even large proportions of code by lead developer, co-developer or establish a quality control before code commit. A number of projects establish mandatory processes for each developer to perform unit tests and reviews before commit. A few respondents replied that paid reviewer from independent projects are invited for reviews.

Most of the projects focus on their test approach and implement a structured test approach, defect handling and test management, which follows e.g. unit testing, functional, integration and regression testing before new versions are released. The maintenance of test frameworks, establishment of test cases or scripts for every new unit seems commonplace. Some projects do live sessions to increase their product quality. Other projects report on performing "smoke" testing of various builds on most supported platforms and configurations, against most commonly used modules. These are followed by unit and functional testing as well as test automating and run with every build. Several projects execute automated nightly tests and maintain checklists for a release. A few projects perform continued risk and ex-

ploratory testing during the development release phase, while full regression testing is performed for a stable release phase. Testing is done manually and automatically, supported by Testing Tools, such as JUnit, which assumes higher importance.

6.2.5.2 QA Actions Performed

With this section, the performed actions by QA teams are analysed, resulting from reviews, interviews or other project tasks. The aim is to examine applied quality assurance activities that are triggered, as a result of observed quality issues.

Several projects report active communication activities, such as group discussion, web meetings or direct mailings to developers to consider defects or requirement changes. Some reconsider their role model and redistribute tasks, as well as planning or design activities. A few projects mention the introduction of a "Quality Team", which acts as an additional layer and certifies changes or the product. The QA team strictly checks the adherence to standards to keep consistency. Other projects involve their engineering team in QA processes, to reduce communication complexity. Improvements of process quality are approached by a few projects, by defining developer guidelines, setting up of overall documentation or reworking coding standards.

Quality problems, resulting from flawed code are addressed with refactoring, which is a known technique from extreme programming. Some projects mention rewriting parts of the code, drive changes for compliance, rework the code to adhere to standards or even reject code with improper syntax or policy violations. Furthermore, code issues have consequences on further releases, such as the reconsideration of scope, definition of release exit criteria to even stopping the release, if problems cannot be solved.

A number of projects enlarge their testing activities and conduct additional code reviews, function tests, recursive tests, intensified testing prior to releases, followed by reporting of inconsistencies found in the project management. In addition, test scripts are enhanced or further test cases added to the existing regression test frame.

Numerous projects improve their defect handling and management processes, as a result of classification and prioritisation of bugs or feature requests, reopening, reprioritizing and assignment of tasks to the development team. Some projects approach repeating defects with self-developed solutions, while a few adopt or even rewrite their testing toolkit.

6.2.5.3 Suggested Quality Improvements

In this section, the participant's suggestions and experiences regarding quality improvement are examined. Open-ended questions were used to capture these findings. The results are

summarised and clustered into the categories of community, knowledge transfer, quality assurance, documentation, design, development, release management, testing and tool support aspects:

- Several respondents see further quality improvements as a result of the enhancement of community aspects, which is a cornerstone of the development model. Projects require more knowledgeable users and developers, who work closely together and are committed to their task. Users need technical skills for defect handling and documentation, but not for coding. Having paid nothing for software, users feel no emotional investment in the product and do not bother to report defects. Therefore, people need to be encouraged to report the bugs they find. Some respondents suggest an increased participation of corporate users, who contribute full-time to OSS projects. Developers need to be educated on how to release or bug fix a branch. The demand for more full-time and even paid professional OSS developers is raised by several participants, as developers contribute mostly part-time to OSS projects without any financial reward. Public recognition of developers could demonstrate higher respect and increase motivation for the project work. An experienced, strong and accessible project management, preferably reporting to a committee, needs to guide development direction, demonstrate leadership and motivate the community. The participant's project roles must be clearly defined as well as the processes. Adequate process documentation supports developers and eases the integration activities for a project.

- Contributors must be committed to deliver the product and spend half their time in coding and half in testing, especially for unit and integration testing. Automated builds and nightly tests could support this. The development team and the Quality Assurance team need to work closely together. Large projects require an independent QA team, to ensure test planning and quality checks beside development. High-level development and quality assessment tools must be available also for an OSS product to ensure common accessibility. The community must be conscious of quality and must question and check the quality of every process as a continuous improvement process.

- Generally, improvements in documentation are seen as being of major importance. Documentation on how to contribute to the project must be available. The developer documentation must contain style and coding conventions and guide developers to comment-based documentation at source code level, to ease sharing, integration or review processes. Adequate help files and end user documentation must be commonplace. In addition, the development order needs to be considered: to document first, then test and then code. Testing documentation must be sufficient and comprehensive.

- Product specifications and requirements as well as usability and portability need to be defined beforehand. A technical design phase, with elaboration of the system architecture must come before the development phase, to avoid shortcomings, fundamental changes or rework of the whole product in the later stages. Engineering should follow international standards or consider the use of a design pattern. Software engineering should be flexible and modular. The selected development approach, such as the "Bazaar way" or "Extreme Programming" must be appropriate for the building of the software product and the concept of reusability needs to be considered.
- Releases need a standard version or number scheme to show differences between feature releases and bug-fix releases clearly. The usage of hybrid release approaches such as mixture between time and feature based release strategy needs to be evaluated. Several respondents mention that more time is required, which indicates too high release frequencies.
- The efficiency of test processes needs to be improved, as a result of more real life user support, a greater adoption of formal testing methods and a 100% code coverage. High user interaction and the involvement of end-user in testing pre-release software are seen as mandatory. Software quality could benefit from more effective code reviews processes, such as inspections, walkthroughs or independent peer reviewing with quality judgements. However, providing enough time for development and testing, still remains an important factor.
- Generally, tools could benefit from higher integration, but should remain vendor independent. Further tools support for dynamic or web applications are desirable. Tools support must be simple, easy to use and should help developers to get started with their tasks as well as providing guidance. Automated test tool should accompany the user in reporting bugs to save developers time. A broader use of automated test suites, supporting e.g. unit or regression tests, should be applied during development, which correspondingly demands the writing of automatic test scripts before or at the same time as the code.

Many assertions have been made in this "free form" participant's feedback, which needs to be carefully assessed and are considered as helpful advice for further research.

6.2.6 Project Success Measures

This section discusses evaluates success according to project success measures as suggested by Crowston *et al.* (2006). The assumption is made that more mature projects are an ideal object of study, as they reduce shortcomings and improve their processes over time. The

premise for successful projects are productive projects, independent of their size (in LOC), that have been available for some time in the market and which are able to attract and manage large communities and development teams. The measurement on project success is an elusive target and depends not least of the beholders position (Kitchenham and Pfleeger 1996). Therefore, measurable variables are used to determine a certain threshold for project success. The project success criteria, such as maturity, time in the market, developer size or community size are defined and the following assumptions are made:

- The project has a productive release version
- The project is more than two years in the market
- The developer team consists of more than five developers
- The community has more than fifty participants

Roughly, one-fifth of the projects correspond to these criteria. In the following, the term "mature projects" is used to describe this group. The analysis covers all projects beyond this threshold and focuses on personal issues, processes and methods, testing, defect handling, QA and organisational issues. An evaluation of general characteristics is ignored, as these projects are mainly filtered according to these criteria. In the further analysis, only major deviations from the previous findings are highlighted:

Personal Issues

The group of mature projects shows a higher level of full-time project participants (S4) (31%), while the level of part-time contribution remains the same (53.6%). Moreover, the participant's motivation (S5) shows increased meeting of company needs (35.7%), which leads to the conclusion that the development of mature projects is often continued for commercial reasons.

Processes and Methods

The analysis of the design, coding and testing activities (P1) shows a slight increase in testing efforts to almost 29% (see figure 75).

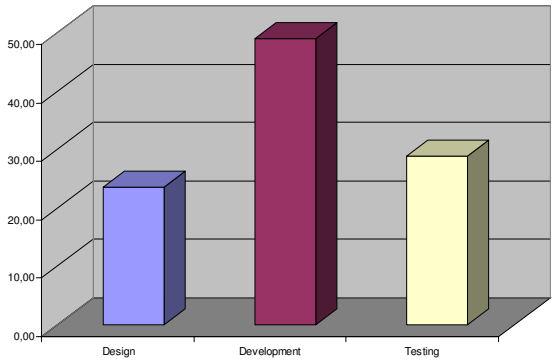

Figure 75. Proportion of Project Activities

The release strategy (P4) is mainly feature-based (66.7%), while there is a clear increase in time-based approaches (17.9%). The majority of projects (39.2%) release (P2) once a quarter, while 32.1% release once half a year. A shift towards a larger release cycle is expected, as the initial development phase is completed and the product exists for some time on the market, but these mature projects show a higher release frequency. The majority of mature projects (44.0%) tend to have code changes (P3) of 10-20% between major releases. The average level of reused code (P5) increases to 36.6%, which is slightly higher than that observed in the whole sample. Code modularity (P6) almost remains unchanged. Over half (54.7%) consider these aspects in the design phase, 13.1% at the start of the development phase and 31% reconsider modularity aspects during development. The average level of abandoned code (P7) is still about 12.9%.

Testing

The proportion of testing efforts (T1) compared to the development time increases slightly to 40.7%. It is noticeable that almost 71.5% follow a structured test (T2) approach with test scripts or sometimes with planning. Users on average test 60.2% of the code (T3), while large projects benefit more from user testing (r_{SP} = .256, p = .019). User suggestions (T4) are more highly rated in mature projects, as 52.3% mention they "bring the design forward" and 42.8% consider them as useful. The user testing efficiency (T5) is remarkably high, as a much higher proportion (77.3%) report their users find hard bugs. Code reviews (T6) are applied more frequently (84.5%) and on average 59% inspect (T7) their source code. The applicability of peer review techniques (T8) increases to 69.5%, as in this group 25% perform peer reviews frequently, 19% on request and 25% for critical parts only. Walkthroughs (T9) are more frequently applied (71.4%), as 17.8% execute them frequently, 40.4% sometimes and 13.1% for special cases only. Mature projects benefit from a higher level of code control. Around 56% of the projects perform corrective actions before code commit (T10), which means that 20.2% reject inappropriate code, while 35.7% rework the code by their lead developer.

Defect Handling

Defect handling (T11) has a higher importance, as 42.8% introduce the process directly from project start, 32.1% start defect handling in the development phase. These projects have an increased defect handling scope (T12). For instance, 95.2% mainly track code defects, followed by documentation (63.1%), requirements (52.4%) and design defects (50.0%). The defect reporting quality (T13) is noticeably high. Around 19.1% claim to receive detailed information, while an increased proportion of 55.9% report using useful bug descriptions. The defect handling process (T14) does not differ, as 48.1% said they assign defects directly

to developers. However, a high number of 76.1% mention having rules for defect classification (T17) and status follow-up. The average defect handling time (T15) is roughly one week, while the majority (41.7%) report it takes longer. It is noticeable that 20.2% of the observed projects have escalation procedures and even 51.1% solve security critical defects (T16) within their whole team.

Organisation and Infrastructure

More than 48.8% of the mature projects have on-boarding procedures (C1), which shows a significantly increased proportion compared to the whole sample. The average time (C2) to take new participants on board is 2-3 weeks. However, 42.8% report they require 4 weeks and more. The communication between user and developer (C3) reports 84.5% as direct and efficient, which is a significant amount.

Documentation has a larger importance in mature projects. For instance, 73.8% have process descriptions (D1), 85.7% have development documentation (D2) and 94.0% mention detailed product documentation (D3) or at least some drafts.

The examination of tool usage (I1) shows a noticeably high usage (see figure 76). Around 96.4% use Source Code Control Tools and more than 90% apply Bug Tracking Tools or Mailing Lists. Web Portals are applied by 66.6%. Both automatic build tools and Testing Tools have a proportion of roughly 58.3%. The use of Instant Messaging Tools seems common, and is applied by 59.5%.

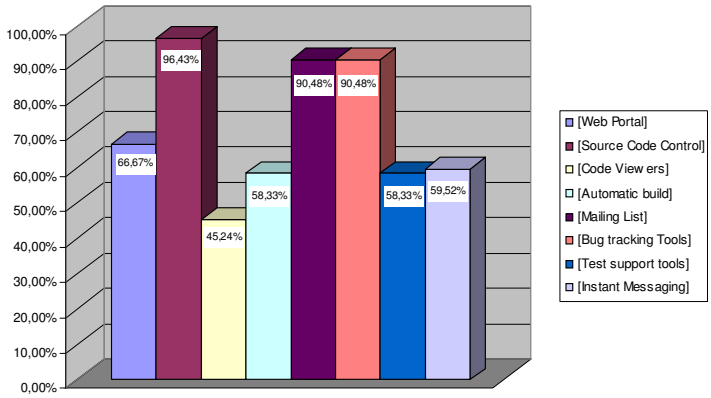

Figure 76. Proportion of Tool Usage according to Project Success Criteria

Quality Assurance

It is noticeable, that 33.3% of the mature projects execute QA practices (Q1), which does not significantly differ from the previous findings. However, 42.8% report that for their QA checks the product adheres to standards and requirements (Q2) and 29.7% of the projects reply that their QA team triggers further actions (Q3), resulting from interviews or inspections.

6.3 Discussion of the Survey Findings

The survey results provide evidence for the lifecycle of the OSSD, as described by Raymond (1999). The existence of large user communities and development groups are found. However, only 40.3% of the projects have user groups of more than 50 people, which confirms the existence of much smaller user groups, as discovered in the study by Zhao and Elbaum (2003). The majority of development teams consist of 2-5 developers, but development teams with more than 20 developers can be found in large projects. Projects tend to start with small communities, as 55% have less then 10 users in the first year, but there is a significant growth of the community size when projects mature. The completion of beta status toward a productive status affects 57.8% of the projects after the first year in the market.

The participant's level of knowledge in software development is relatively high, as more than 75% claim to have more than five years of experience. An experienced team is considered as an important element, as professional developers provide accurate feedback (Massey 2003) and may explain the high product quality that OSS projects can deliver. The knowledge transfer to new participants lasts on average 2-3 weeks. Only mini projects report an on-boarding time of about one week, while large projects need 4-5 weeks and more. Hence, knowledge transfer procedures are more frequent in larger projects. The average team size is 2-5 developers and participants adopt multiple roles. Developers are motivated mainly due to personal needs in mini to mid-sized OSS projects, while company needs are more crucial in large projects. This corresponds with the study of Hars and Ou (2001), who observe an external personal motivation of developers. More than 54% of the participants work part-time for OSS projects. However, large projects show a high proportion of full-time participants, which corresponds to an increased company motivation. Projects seem to have stable organisations with less fluctuation, in contrast to the assumption of volunteers freely leaving or joining. Mature projects, which are more than four years on the market, report less than 1% of new participants with minor development experience. Thus, participants attend projects over a long period, as a high degree of personal motivation exists, when developers for example are users of the product.

The analysis of the design, development and testing activities shows a uniformly distributed picture in relation to project size. With project growth, the proportion of development time increases to the disadvantage of testing activities. Mature projects especially seem to spend more time in testing.

Most of the projects follow a feature based release strategy, which is triggered by the readiness of the code. The time-based approach has a higher importance in mature projects and contributes to a more sustainable development process (Michlmayr 2007). Furthermore, hy-

brid approaches as combination of time and feature-based strategies could be cumulatively observed.

The release frequency is lower than expected, as the majority of projects (32.7%) release twice a year. However, most of the mature projects have rather shorter intervals, which corresponds to Raymond's statement "release early, release often". The majority of projects (38.9%) have source code changes of about 10-20% between major releases, which shows a manageable complexity, while a smaller group report changes of more than fifty percent of their code. On average 24.3% of the code is reused, which is an inherent element of the OSS development. Mature projects utilize highly code reusability and report an average usage of 36.6%. Branches, which are up to 100% clones, might show the result upwards. Mature projects consider code modularity more frequently during the design of the product. Aspects, such as the high modularity and the system design prior to the development largely contribute to software quality (Aberdour 2007). The examination of the level of abandoned code shows an estimated value of an average 12-13% across both analyses. Projects with an increased level of abandoned code report more quality issues and side effects than other projects. This corresponds to the findings of Michlmayr *et al.* (2005), who observe certain quality issues due to unsupported code.

The mean proportion of testing time in relation to development effort is 38.6% and reflects the findings of Zhang and Pham (2000). More than half of the projects (52.3%) follow a structured testing approach and report a high efficiency of user testing (57%). It is noticeable that 71.5% of the mature projects have a structured testing approach and significantly benefit from user testing, as on average more code is tested with a more efficient reporting quality and user suggestions are more efficient. The remarkable communication between developers and users contributes to this effect. Hence, mature projects use their communities more efficiently for testing and debugging activities and significantly benefit from this approach. A well-organised approach delivers better results because users have more guidance and clear processes. User testing constitutes one of the most important QA elements in the OSSD model. A comparison of review techniques to user testing shows that projects typically seem to benefit more from user testing, which emphasises again its impact. Overall, OSS projects frequently apply code reviews (69.9%), walkthroughs (61.4%) peer reviews (50.5%) and typically inspect 47.2% of the code. Review techniques, such as code reviews, walkthroughs and peer reviews are more frequently applied in the group of mature projects with an even higher degree of inspected code (59%). This confirms Raymond's (2001) assumption that "the high level of quality is partly due to the high degree of peer reviews and user involvement". In addition to testing techniques, 46% of the projects perform corrective actions before the contributed code is committed to the repository. The study shows that large commu-

nities have strict quality checks, as 75% of this group report reworking or even rejecting inappropriate code.

Mature projects introduce defect-handling processes mainly at project start or at start of the development phase and have a much wider scope. The majority of projects (88.9%) track source code defects, while half of the projects typically consider requirements, design or documentation issues. More than 50% receive useful bug descriptions, while this value significantly increases in the group of mature projects (75%). The study shows an advanced level of reporting quality, which is contrary to Michlmayr *et al.* (2005), who argue that developers see an increase of useless bug reporting with less technically skilled users. The results confirm that distinctive processes, such as a structured approach or well-documented processes contribute to a high reporting quality. More than 77% report having classified defect-handling descriptions. This study demonstrates an improved situation in contrast to Villa (2003), who observed lack of information about priorities or degree of severity for defect handling. The average defect handling time is about one week, while a majority of projects require even longer intervals. This finding supports the position of Michlmayr *et al.* (2005), who see quality problems regarding reliability or response time. In particular, the handling of security critical defects is regarded as an issue, as only 8.9% report having procedures to react to them, while 28.8% have no special treatments and even 10.6% seem to be unable to determine the defect criticality. If projects have problems in determining the criticality of their defects, this has serious impacts for the maintenance, as appropriate measures to counteract cannot be coordinated in time. It is noticeable that mature projects have a higher focus on security critical issues, as around 20% have defined escalation procedures with assigned responsibilities.

Knowledge transfer processes are more frequent in larger projects, and especially mature projects force their implementation. Similarly, the average time for knowledge transfers increases with project size, which indicates the complexity of the projects. The efficient management of knowledge transfer has, especially in larger projects, an increased importance in reducing the on-boarding time, while increasing the developer's efficiency. The precondition for this is an appropriate process and product documentation. In general, process descriptions seem to be underused (30.1%). Roughly, two-thirds have coding and development style guidelines or product documentation, while mature projects have a much higher focus on it. For instance, process documentation is generally available and contributes to knowledge transfer processes, while almost every project has product documentation or at least a draft. These findings confirm an improved situation, compared to Michlmayr *et al.* (2005), who observe a lack of user and developer documentation. The communication problems in medium or large size projects, as observed by Michlmayr *et al.* (2005) cannot be found. The

participants rate the communication between developers and users as efficient. Larger projects communicate more effectively than smaller ones, which contradict Michlmayr's findings.

OSS projects make significant use of tools. Source code control tools, bug-tracking tools and mailing lists are frequently applied, while testing tools are underused. Projects that apply bug-tracking tools report typically a higher defect reporting quality. Tool usage correlates with the project size, because large projects make more use of supporting tools, than smaller ones. For instance, 60% of the mini projects do not apply "Bug-tracking Tools". Tools offer scalable solutions for the collaborative development approach to counteract increased complexity.

Only one-third of the projects apply Quality Assurance practices. However, there is a significant trend in larger projects. The study shows that several projects discuss their quality issues within the core development team or their community. Processes and standards are defined to which each community member needs to adhere. Some projects reviewed the structure of their project organisation and set up QA teams, who contribute full-time to the project. Several projects frequently conducted different kinds of code reviews and implemented structured test approaches, defect handling as well as test management. Several projects established live sessions as well as automated nightly testing to improve software quality. Therefore, the usage of "testing" tools becomes more important.

Only one-fifth of the projects responded that their QA team perform actions, such as group discussions, mailings or even organisational changes of their QA team that verify processes, documents and checks the adherence to standard. Other projects adopt their development approach and apply "Refactoring" processes or rework flawed code in case of quality problems. Quality issues can have several impacts on further releases as some projects reconsidered scope, defined exit criteria or even stopped the release. Several projects introduce additional code reviews and increase their functional or regression tests prior to releases, while many projects report further improvements of their defect handling and bug reporting processes.

The respondents' suggestions for quality improvements show that the enhancement of community is one major aspect, such as knowledgeable users, professional developers contributing full-time to projects, high integration of users and developers and even paid contributors (Otte *et al.*, 2008a). In accordance with the level of understanding by the community, an experienced project management needs to provide leadership and guidance for their projects. Large projects require an independent QA team (Otte *et al.*, 2008a). However, the whole community must be conscious of quality to achieve significant improvements. QA ought to

perform checks supported by QA tools to verify that processes or guidelines are kept. OSS projects should provide process documentation and focus on development guidelines, comprising style or coding standards. A technical design phase prior to the development could reduce shortcomings in architecture or development approach (Otte *et al.*, 2008a). An appropriate release approach must be defined, regarding strategy, frequency and scope. Otte *et al.* (2008a) conclude an important area for improvements are testing processes, such as test efficiency, improvements of code reviews or the use of testing tools for automated testing. Tools should be OSS products themselves to lower barriers to communities and should have a high focus on integration, while there is further demand for easy handling and simplicity (Otte *et al.*, 2008a).

6.4 Conclusion

The survey research analyses quality assurance practices and focuses on key practices in mature projects. The results provide evidence for the OSSD lifecycle described by Raymond. Projects follow frequent releases, have a high user involvement, benefit significantly from user testing and peer reviews. There is a significant growth of communities when projects mature and the existence of large user communities and developer groups is observed. Developers mainly contribute for personal or community needs, however a higher commercial motivation exists in large projects. In general, an excellent level of development knowledge can be observed. This leads to the assumption, that professionals know how to approach things 'right', which could explain the large number of successful projects collaborating in a loose manner sometimes with informal processes and fewer rules (Otte *et al.*, 2008a).

An investigation into more mature projects provides important insights into applied practices and may indicate reasons for the success of the OSSD model as follows. Otte *et al.* (2008a, p.1252) conclude:

> "These projects consider modularity of code already during design, which shows the concern to base the development on a solid architecture. Quality control activities before code commit have a higher importance. More time is spent on testing and the testing approach is better structured. Mature projects efficiently leverage their community and benefit from efficient user testing. Internal communication is rated as remarkable, which contributes positively to these processes. Defect handling processes seem to be better structured and include source code defects, requirements and documentation issues. Documentation has a higher significance. Mature projects emphasise widespread documentation and use this information to support the knowledge transfers to developers and users. The tool usage is high, making additional use

> of instant messaging, bug tracking tools and test support tools. The establishment of QA processes seem to play a more important role in larger projects and are underdeveloped in smaller ones."

In conclusion, the survey findings show evidence of applied quality assurance processes in OSSD projects. Furthermore, the analysis of "successful" projects demonstrates applied key processes in mature projects, which the OSS project may use to improve their processes (Otte *et al.*, 2008a). Otte *et al.* (2008a) find the relation between success criteria indicates some correlations between quality and success but no causalities. The correlation of QA practices to project success needs to be explored by further research.

The presented survey results provide empirical evidence of applied key processes in mature projects. The important findings that contribute to the development of the QA framework are marked as follows in squared brackets and will be carried on by further research (refer to chapter 7.3.3). In summary the main findings are:

- The results provide evidence for the lifecycle described by Raymond [1]
- Large user communities and developer groups can be observed [2]
- Projects have a significant growth of their community size when they mature [3]
- Large projects consider more frequently knowledge transfer procedures [4]
- Software development teams have a high level of knowledge and experience [5]
- A high proportion of external motivation of developers could be observed
- Internal communication show a high efficiency even in large projects [6]
- Code modularity finds early consideration during design [7]
- Typically a quarter of the code is reused within projects [8]
- Projects perform measures to strictly control their code quality [9]
- A proportion of typically twelve percent of the code is abandoned
- A feature based release strategy is mostly applied, while there is an increased level of time-based approach in mature projects [10]
- Evidence of significant user testing with high efficiency is observed [11]
- Structured defect handling frequently applied with the main focus on source code [12]
- Review techniques, such as inspections or peer reviews are frequently applied [13]
- Relation of development to testing increases with projects growth [14]
- Projects in general have coding or style guidelines, while mature projects focus more on process documentation [15]
- Tools are highly used, especially in mature projects [16]
- Larger projects have a stronger focus on Quality Assurance processes [17]

Part 3 - Framework Refinement, Experiment and Discussion

Part 3 describes the development of a QA framework and its examination in practical projects. The refinement of the model is discussed and closes with the research conclusions.

A generic model is introduced and the taxonomy of the processes and the framework are described. The findings from the previous part contribute to the development of the QA framework that comprises process and product quality characteristics and suggests best practices.

A case study approach is selected to examine the framework applicability in actual OSS projects. The different cases are explored and an interpretative and statistical analysis is performed to complete the research.

The case studies' findings contribute to the validation of the model and a critical evaluation of research limitations. The thesis closes with discussion of the key findings, shows their contribution to knowledge and states areas for further research.

7 Quality Assurance Framework for the OSSD Model

This chapter concerns the development of an original Quality Assurance Framework to support the OSS development lifecycle. An underlying research approach is established, which provides the structure for further evaluation. A summary of the recent findings is provided and the need for a comprehensive model is shown. The research approach, the aims and objectives are explained. A framework, which focuses on quality assurance process in the OSS development lifecycle, is developed.

7.1 Research Focus

The second research question, posed in chapter 1.1, focuses on two major aspects.

> **Second Research Question:**
>
> *"Can a quality assurance framework contribute to the improvement of the quality assurance activities in the OSSD?"*

First, it implies the development of an underlying approach and thus the establishment of a QA framework. Second, it concentrates on the exploration of its contribution value, which requires an in-depth study of the model in real situations.

This chapter examines the first aspect that could be formulated as:

> *"How can a quality assurance framework for the OSSD be established?"*

This leads to subsequent questions:

- What are the characteristics of the OSSD that affect software quality?
- What are the interests of the stakeholders?
- Which main processes are relevant for quality assurance?
- How do they differ to standard models?
- How could an appropriate process model be established?
- Which best practices could enrich the model?
- What kind of quality measurements can be adopted?
- What kind of determination methods could be applied?

7.2 Research Approach

The proposed development of the framework follows an approach that comprises four major steps:

- Define model fundamentals
  - Specification of the model aims, objectives and deliverables
  - Analysis of the OSSD quality characteristics
  - Identification of the project requirements and stakeholder quality views
  - Review of existing models and approaches

- Specify model components
  - Introduction of a generic model approach
  - Definition of the OSSD quality characteristics
  - Establishment of the model components and attributes
- Define measures
  - Definition of metrics
  - Development of a measurement model
  - Definition of appropriate reference values for measurable attributes
- Collate the model
  - Consolidation of the framework elements
  - Implementation approach for the model

First, the characteristics of the OSSD model are analysed. Therefore, features and observed QA practices are shown. A summary of the strengths and shortcomings in the OSSD to assure software quality is given and the need for a process model is emphasised. An analysis of the model requirements is provided and the different views of the stakeholders are shown. The aims and objectives of the framework are defined and an overview of the expected outcome is given. Finally, the existing QA approaches are reviewed and the necessity for the development of a QA process model is shown.

Second, the process model components are developed. A generic process framework is introduced to establish the underlying structure of the process model. The process model describes the components and their attributes.

Third, the measurement model is established. The purpose and aims are defined and the different measurement objectives, such as process capability, product quality and project success factors are examined. A determination method for each measurement approach is introduced.

Fourth, the framework is consolidated. The dependencies and impacts of the elements are shown and a QA framework is designed. The underlying assumption of the QA framework is discussed and a recommendation for its implementation is given.

7.3 Model Premises and Fundamentals

7.3.1 Aims and Objectives

The following aims and objectives can be defined:

Aims

- Establishment of a process and measurement model with focus on quality assurance and control
- Provision of an OSSD process model concerning QA relevant processes

- Definition of a measurement approach with metrics and a determination method
- Improvement of process efficiency, to support the management of existing approaches and to ease the set-up of new projects

Objectives

- The framework focuses on simplicity and ease of use
- A generic approach is defined as basis for the process model
- The process model comprises quality relevant processes and is lifecycle or method independent
- The QA framework focuses on the developer view (product revision, transition) and user view (functionality, stability, etc.)
- The target groups are mid-size to large projects

Deliverables

- A process framework that supports QA practices in the OSSD
- A measurement model, which comprises process quality, product quality and project success assessment with a respective determination approach
- A process capability determination approach for the assessment of the process model
- A project success capability determination approach for the project success metrics model

7.3.2 Stakeholder Quality Expectations

Different quality expectations of the stakeholder need to be considered for the model alignment. The different roles, as introduced in chapter 3.4.1, distinguish various motivations or quality views. In theory, the ideal quality views are as follows:

- Users are mainly motivated by personal needs and strongly focused on the application of the product. Their quality expectations comprise internal and external quality criteria, such as functionality, usability, integrity, efficiency, correctness or reliability of the product.

- Developers contribute to the product design mainly for personal and company needs. These come from personal interests through using the developed product or an intrinsic motivation, which is rewarded by the community. Their quality expectations comprise internal and external criteria, with attention to maintainability, testability, flexibility, reusability or portability.

- Committers belong to the group of developers with extended rights to commit code to the repository and to perform quality control or code reviews. Thus, committers actively perform quality control of internal and external quality criteria.

- Release managers are developers with access to the repository. They focus on release management, such as scoping, incorporation of patches, scheduling or execution of test-

ing phases. Although, their motivation is similar to developers', they have a strong focus on verification and validation aspects of the product.

- The group of project managers consists of developers or committers, who have a high personal or commercial motivation to force the project. Their focus is the management and control of activities, such as quality or performance measurement.

This research follows the quality criteria, as discussed in chapter 2.6 and incorporates two main views that need to be reflected in the quality assurance framework. The user view is considered to focus on product operational issues. The developers view takes product revision, or transition criteria into account.

7.3.3 Proposed Key Processes of the Quality Assurance Framework

The recent findings from literature and survey research, which were detailed in chapter 6.3, contribute to the proposition of the key processes of the quality assurance framework.

The high quality in OSS relies on large communities, which enable effective debugging, user testing and suggestion of new features, high code modularity, frequent release cycles, application of peer reviews as well as an organized environment supported with tools (Aberdour 2007). Recent research has shown that "tool usage and user participation is high, which effectively supports testing; defect handling is well structured, but documentation is often rare" (Otte *et al.*, 2008b, p.124). More mature projects consider code modularity during software design. These projects have stricter quality control activities before code commit and spend more time on testing (Otte *et al.*, 2008b). Furthermore, defect-handling processes are well structured, documentation has a major importance and supports the knowledge transfer. Internal communication seems remarkable; organisations are well defined, highly supported by tools and SQA processes are more often implemented in larger projects (Otte *et al.*, 2008b). According to recent findings, as noted in chapter 6.4, the following key processes are suggested:

- **Product and Process Documentation ([15])**
  The establishment of proper documentation has a high impact on quality, as its lack complicates the knowledge transfer to new participants (Michlmayr *et al.*, 2005). Furthermore, documentation is essential for usability and knowledge capture.
- **Coordination and Team Communication ([3], [6])**
  Project success depends on the establishment of a highly organized approach (Aberdour 2007) and the establishment of an effective organisation becomes a vital issue to produce high quality software (Otte *et al.*, 2008b). Otte *et al.* (2008b, p.125) conclude: "appropriate structures are required to avoid frustration and to keep people

constantly motivated. This includes effective coordination of project tasks by the core development or management team. Efficient communication plays an important role in a geographically distributed development approach. It constitutes the basis for collaborative development processes and enables the frequent distribution of information".

- **Infrastructure and Tools ([16])**

  The infrastructure directly affects the project co-ordination and communication (Michlmayr *et al.*, 2005). The establishment of appropriate tools for the infrastructure has a significant impact on process quality and collaboration (Aberdour 2007). The integration of tools offers a broad range and constitutes the basis for a collaborative distributed development approach with manageable complexity (Otte *et al.*, 2008b).

- **Knowledge Transfer ([4], [5])**

  Knowledge transfer processes gain an enormous importance for software quality to avoid a lack of expertise, as developers loosely collaborate on a voluntary basis (Otte *et al.*, 2008b). Recent findings show that knowledge transfer processes are an important element to support new developers. Otte *et al.* (2008b, p.125) find, "appropriate process documentation eases the knowledge transfer for new developers and can be used to counteract the varying skill levels within the community."

- **Requirements Evaluation and Design Reviews ([7])**

  Villa (2003) emphasises the need for initial planning and detailed documentation of design documents. An imperfect design phase that lacks system analysis and the definition of the system architecture prior to coding, might lead to severe quality issues when requirements evolve (Otte *et al.*, 2008b). The focus on system design is an important criterion to achieve high quality (Aberdour 2007)

- **Software Engineering Control ([8], [9])**

  Software Engineering Control processes, such as the adherence to coding standards and guidelines, constantly reviewing changes, performing of corrective actions, rejecting inappropriate code and peer reviewing by their community are important quality criteria. Otte *et al.* (2008b) conclude that quality control processes before commit are considered an important element for a constant code quality control.

- **Peer Reviews and Inspections ([1], [2], [13])**

  Reviews and inspections processes have a major impact on software quality and are considered a key element for quality assurance in OSS projects. These processes are parallelisable and scale up against system complexity (Raymond 2001). The more people look at the code, the higher the ability to detect defects early. Hence, the ap-

plication of reviews and inspections in OSS become an integral element (Dinkelacker *et al.*, 2002).

- **Verification and Validation (Testing and Defect Handling) ([11], [12], [14])**
  Larger OSS projects take the advantage of their community size and benefit from user testing with high efficiency (Otte *et al.*, 2008b). OSS projects have varying testing strategies; some projects require unit tests with every committed code, while others use automated testing and set up a test suite. Otte *et al.* (2008b) conclude, a structured testing approach, where integration or regression testing is performed before major releases, is an important criterion to delivery high quality software. A solid test management strategy needs to be established to support and to guide testing activities. Moreover, a perfect bug-reporting quality is an important success criteria for defect-handling processes (Michlmayr *et al*., 2005).
- **Release Management ([10])**
  The proper release management and strategy have an important impact on software quality (Porter *et al.*, 2006). Michlmayr (2007) has shown the quality improvements from time-based release approaches and its efficiency as a result of improved planning and control mechanisms.
- **Software Quality Assurance ([17])**
  The definition and applicability of QA processes across OSS projects is not prevalent, as projects often perform SQA tasks informally without any strategy or planning (Otte *et al.*, 2008b). In this research, the SQA process is considered as a frame, which combines relevant QA processes, forces their establishment and controls their achievements.
- **Quality Management ([17])**
  Otte *et al.,* (2008b, p.126) find that "OSS projects apply quality targets and the respective assurance processes are mostly informal." In addition to some benchmarks or code coverage targets, a strict quality management approach, which includes the definition of quality goals and their assurance is considered as important software quality criteria.
- **Continuous Process Improvement ([17])**
  Projects do not explicitly define the continuous improvement process or state its execution, nor does the qualitative analysis of QA practices indicate its existence (Otte *et al.*, 2008b). Otte *et al.* (2008b, p.126) conclude, "the spirit for continuous improvement implicitly exists within communities and developers feel empowered expressing their ideas for improvement". Thus, this process is considered a key element for a quality assurance framework.

7.4 Conceptual Framework Design

Rather than directly assuring software quality, the SQA framework needs to provide an adequate assurance that projects fulfil their product and quality targets. The achievement of a high product quality is seen as being dependent on the process used to build it (Baker and Fisher 1999). The entire solution reflects a process model, a process assessment and a success measurement approach that are consolidated in the Quality Assurance Framework for Open Source Software (QAfOSS). The process model is a "blueprint of how to organise, implement, conduct, and manage software engineering processes in an organisation" (Wang and King, 2000, p.52). Hence, a quality assurance process model describes perspectives, orientation and deliverables of quality assurance processes and methods. The establishment of a quality assurance process model requires the definition of an underlying process framework and a subsequent process structure. First, a hierarchical process structure is developed which describes the process targets, outcome and activities. Second, a generic process framework is introduced, which defines the taxonomy of the process model.

7.4.1 Fundamental Process Structure

The ISO 12207 (2007, p.5) defines a process as a "set of interrelated or interacting activities which transforms inputs into outputs". According to IEEE-STD-610 in Paulk *et al.* (1993, p.3) a software process is defined as "a set of activities, methods, practices, and transformations that people use to develop and maintain software and the associated products." Following this definition, the structured process description of the CMM (Paulk *et al.*, 1993, p.29) is applied as starting point. The proposed taxonomy of processes follows a hierarchical structure and consists of process groups, features and practices. The aim is to provide a process structure showing process groups and subsequent processes, as depicted in figure 77. The highest node is defined as process group and adopts the definition of a process category, which "is a set of processes that are functionally coherent and reusable in an aspect of software engineering" (Wang and King, 2000, p.52). Each process group can be assigned to a maturity level, indicating the process capability similar to the CMM. A process describes a set of sequential practices for software project organisation, implementation, and management (Wang and King 2000). Processes have a purpose and an outcome and consist of certain practices. Each practice describes activities that contribute to the implementation of the process. Wang and King (2000, p.52) define "a practice as an activity or a state in a software engineering process that carries out a specific task of the process".

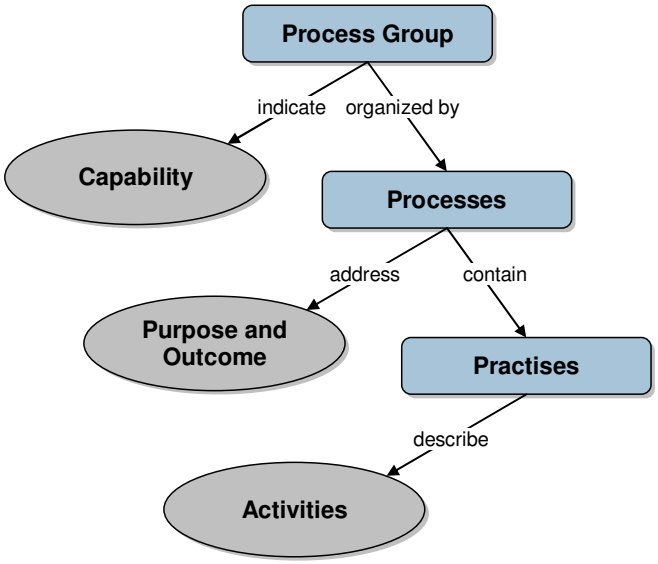

Figure 77. Revised Generic Process Structure

7.4.2 Generic Process Framework

The generic process framework defines the taxonomy of the process model. It distinguishes process objectives, characteristics and evaluation criteria. The process objective reflects the process domain, which is "a set of ranges of functional coverage that a process model specifies at different levels of the process taxonomy" (Wang and King, 2000, p.53). The authors classify the three process domains organisation, development and management. In addition to these three domains, which are adopted, the domain "resources" is introduced in order to show a detailed partition of organisational and infrastructural aspects. Hence, the proposed process framework distinguishes the following process objectives:

- **Management** clusters all processes that have a management perspective
- **Organisation** focuses on all processes with an administrative purpose
- **Resources** contain all processes that affect directly the resources
- **Development** concentrates on processes that focus on the product and the development process

The process groups are clustered according to their objectives to a process domain. The process characteristics reflect the process structures and define the practices. The practices describe activities that contribute to the process fulfilment and constitute the lowest level of the model. The evaluation of the practices requires the definition of measurement criteria. They allow the assessment of the implementation of the practices within the development lifecycle. The measurement approach is introduced in section 7.6. The proposed generic process framework, as depicted in figure 78 represents the overall structure of the process model.

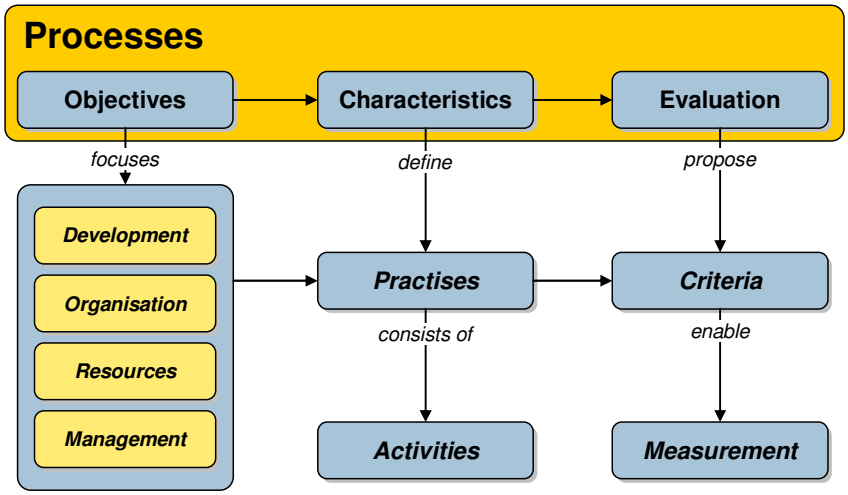

Figure 78. Generic Process Framework (Otte *et al.*, 2008b)

7.5 Proposed Process Model

The process model describes quality assurance processes independent of a development method. It can be adapted by organisations and tailored to their needs. A set of common processes is defined that constitute a reference model. A process reference model is "an established, validated, and proven software engineering process model that consists of a comprehensive set of software processes and reflects the benchmarked best practices in the software industry" say Wang and King (2000, p.50). These "best practices" originate from international standards, such as ISO 12207 or established models, such as CMM. Respective quality assurance processes in the OSSD lifecycle are adopted to build the process model. The scope of the model comprises main QA activities and dependent, preliminary processes.

The proposed process model reflects the dimensions "development lifecycle" and "process domain". The first dimension indicates the initial introduction of each process group in the development lifecycle. Initialised processes may be continued in later stages of the lifecycle, especially due to the iterative character of the OSSD approach. The dimension "objectives" reflects the process domain accordingly. All processes are clustered into process groups and as defined by this research can be arranged according to the matrix structure as illustrated in figure 79. As defined in the generic process structure, each process group consists of subsequent processes, which are described by their purpose, target and practices.

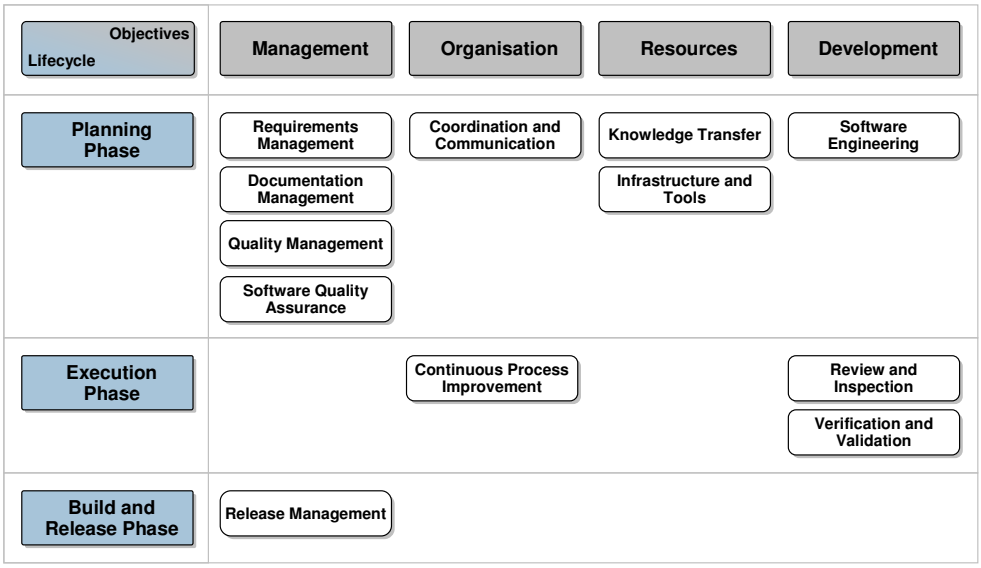

Figure 79. Proposed Quality Assurance Process Model

7.5.1 Management Processes

7.5.1.1 Requirements Management

The proposed process group "Requirements Management" focuses on capturing, management, control and review of project requirements. An extended focus on continuous requirements evaluation under the OSSD is required to ensure a permanent reflection of the actual needs. The correct capturing of the requirements is a basis for the definition of product quality characteristics and the acceptance of the product in the market.

Process	**Requirement Capturing**
Purpose	Management of product requirements
Outcome	Capturing and documentation of product requirements ensured; Process for requirements management established
Practices	Continuous requirements capturing; Perform requirements analysis; Establish documentation and tracking of requirements; Establish change management processes; Introduce a requirement management tool; Record changes

Process	**Requirements Review**
Purpose	Review product requirements
Outcome	Product requirements are documented and up to date
Practices	Conduct requirements review; Incorporate changes into project planning; Review changes

7.5.1.2 Documentation Management

This group is proposed to focus on the product and process documentation. Process documentation describes mainly internal project processes, such as the development approach and applied standards, while product documentation comprises architecture or design specifications as well as user documentation or training materials.

Process	**Process Documentation**
Purpose	Ensure documentation of processes, standards and templates; Establish collaborative accessibility to documentation
Outcome	Collaboration and software development processes are described; Development style and coding guidelines are available
Practices	Describe development processes, rules and guidelines; Prepare development style and coding guidelines; Develop templates to standardise development activities; Continuously review process documentation

Process	**Product Documentation**
Purpose	Ensure adequate product documentation; Establish collaborative accessibility to documentation to ease collaboration and knowledge transfer
Outcome	Description of software engineering relevant documentation regarding requirements, specification, design, architecture, configuration; Description of user relevant documentation regarding functionality, handling and maintainability of the product
Practices	Plan and schedule required documentation; Prepare documentation according to plan; Perform reviews

7.5.1.3 Quality Management

The proposed Quality Management process concentrates on the establishment and management of quality goals and triggers the product quality measurement activities. The definition of product quality goals and the tailoring of appropriate measurement criteria to project needs are considered major parts of the SQA framework. The process follows the CMM standard.

Process	**Quality Management**
Purpose	Product or processes fulfil the quality objectives
Outcome	Quality management procedures are defined; Quality targets are assured

Practices	Define quality targets and standards; Establish plans to achieve the quality targets; Review quality targets and revise them; Measure product quality; Perform corrective actions

7.5.1.4 Software Quality Assurance

Software Quality Assurance processes are considered as mandatory practices a project needs to fulfil to assure the process and product quality within the entire development lifecycle. The proposed SQA process group follows the ISO 12207 standard and is revised.

Process	**Software Quality Assurance**
Purpose	Assurance of product and process quality according to plan
Outcome	Quality Assurance strategy is defined; SQA activities are planned and performed; Adherence of products and activities to the applicable standards, procedures, and requirements are verified objectively; SQA activities/approaches are communicated to the project team; Issues are addressed to the management
Practices	Define QA strategy or plan; Defines standards and procedures; Setup QA tasks and plan; execution of activities; Perform reviews to verify the adherence to standards; Identify deviations from plan; Perform measurements; Report to team and management;

7.5.1.5 Release Management

The proposed process group “Release Management” comprises the management, control and build of product releases. The focus of release management processes is the definition of the strategy, scoping and scheduling to deliver high software quality in a new release. An inappropriate management, scheduling or too wide scoping might deliver incomplete, unstable or inadequately verified products. Therefore, release management requires the completion of verification and validation processes. The build and release check process focus on a pre- and post release quality control. The early delivery of a pre-final release “candidate” allows collaborative testing of the actual version to verify its stability, until the final release is built.

Process	**Release Strategy**
Purpose	Management of release scope and scheduling to ensure stable releases
Outcome	Adequate release management strategy which reflects the development progress, the product maturity, requirements changes, scope enhancements or defect corrections
Practices	Define release strategy (time and/or feature based); Reflect product maturity and perform release quality check; Identify requirements changes and scope enhancements; Incorporate relevant corrections; Verify release stability

Process	**Build and Release Check**
Purpose	Verification of requirements and release control
Outcome	Establish quality control to ensure defined product quality
Practices	Perform pre- and post release quality control; Perform verification and validation processes; Distribute a pre-final release for collaborative testing; Ensure compliance with product quality goals; Verify compliance of product documentation

7.5.2 Organisational Processes

7.5.2.1 Coordination and Team Communication

The proposed process group “Coordination and Team Communication” concentrates on the intergroup exchange of information. The establishment of an effective communication is regarded as an essential element in a distributed development environment to spread information within the community. It also ensures daily communication in the development process. The coordination perspective focuses on task tracking and control of results. It presupposes the establishment of a suitable project organisation with roles and hierarchies, as discussed in chapter 3.4.1.

Process	**Project Organisation**
Purpose	Ensure effective organisational structures
Outcome	Organisational structures are defined and established
Practices	Define organisational project structure with an appropriate hierarchy; Setup communication and collaboration guidelines; Establish OSSD role model and define authorisations; Tailor role model according to project needs; Review and adjust model structure

Process	**Project Coordination**
Purpose	Coordination of project tasks is established and results are controlled
Outcome	Project tasks, issues, work packages are assigned to team members; Results are checked and tracked
Practices	Establish and document coordination rules; Define work packages / tasks for software engineering activities; Ensure accessibility to all tasks for all participants; Force collaborative tracking of accepted tasks; Ensure open information and discussion about tasks; Check and review results by participants and core team

Process	**Team Communication**
Purpose	Ensure effective team communication and exchange of information within the community
Outcome	Rules, guidelines and infrastructure for communication is defined and established
Practices	Define communication rules and guidelines; Establish appropriate communication tools (mailing lists and instant messaging); Establish proper communication cycle (recurring tasks)

7.5.2.2 Continuous Process Improvement

The proposed continuous improvement processes reflect the capability of a team to transform a continuous improvement approach into the team behaviour and the willingness to critically question processes. Such a mindset may contribute positively to the defect prevention approach. The ability of an organisation to reflect critically on their current practices is considered as mandatory in mature organisations. The subsequent processes, like the "Process Change Management" and "Defect Prevention" are adopted from the CMM model.

Process	**Process Change Management**
Purpose	Continuous improvement of the software development process; Improvement of product and process quality
Outcome	Continuous improvement process is established community-wide; Processes are continuously improved
Practices	Define improvements targets; Empower the team to improve processes; Establish an improvement process within the community; Evaluate and implement identified improvements; Perform team training and knowledge management

Process	**Defect Prevention**
Purpose	Identification of cause of defects, prevention of recurring defects
Outcome	Preventative measurements to avoid defects; Common causes identified and systematically eliminated
Practices	Defect prevention activities are planned; Establishment of defect prevention process within the community; Identification of causal relations; Adaptation of processes and standards; Record and review defect prevention activities

7.5.3 Resource Related Processes

7.5.3.1 Knowledge Transfer

Knowledge transfer processes are proposed to focus on educational and knowledge capturing activities within a project. The ability to effectively transfer know-how is a key element to counteract the lack of expertise due to fluctuation. An efficient knowledge transfer process supports the taking on of new developers and is a foundation for team education. The establishment of knowledge transfer processes increases the capability of an organisation to ensure knowledge about e.g. product, processes, methods or the development approach.

Process	**Knowledge Capturing**
Purpose	Counteract lack of project know-how due to fluctuation
Outcome	Knowledge management and documentation procedures
Practices	Ensure up-to-dateness of product and process documentation; Ensure documentation of source code; Ensure documentation of each work package before commit; Continuous review of documentation

Process	**Team Education**
Purpose	Improvement of varying skill levels within team; Preventative measure to avoid defects and shortcomings
Outcome	Professional knowledge about processes, approach, methods, rules, guidelines, standards, etc. across the team ensured; Application of a professional development approach; Training activities planned and conducted
Practices	Analyse/monitor development skill level; Prepare training or knowledge transfer documentation; Conduct team training or ensure education through knowledge transfer documents; Continuous review of team skill level

7.5.3.2 Infrastructure and Tools

The proposed process group "Infrastructure and Tools" relates to the proper development environment and tools, which support the processes, guidelines and standards. The definition and set-up of a common project infrastructure, consisting of development and collaborative tools, must reflect the respective development approach. While collaboration tools implement such processes, improper tools can hinder the development process. Especially in the distributed development environment, instant messaging techniques offer advantages for collaboration.

Process	**Infrastructure Management**
Purpose	Define tools for entire lifecycle, which support the collaborative distributed development approach and reflect or enable project processes
Outcome	Software engineering processes effectively supported; Collaboration processes supported; Development processes reflected and supported
Practices	Establishment of a freely accessible web portal; Implementation of a source code control tool; Recommendation of code viewers; Setup of an automatic build tool, if appropriate; Introduce a bug tracking tool; Introduce a test support tool; Initialise collaboration tools (mailing list, instant messaging); Describe guidelines and document project tool standards; Review the effectiveness of the tool support

7.5.4 Development Processes

7.5.4.1 Software Engineering

The proposed process group "Software Engineering" comprises the software development activities concentrating on design and source code development. The "Requirements Management" is a precondition for software engineering. The design process describes all activities transforming requirements into the functional and/or technical specification. Source coding builds upon the design process and describes the product development, such as source code or documentation. The process group requires an appropriate team communication, infrastructure and triggers the "Review and Inspections" and "Verification and Validation" processes.

Process	**Design Control**
Purpose	Establishment of functional and technical product specification, which can be verified against the requirements
Outcome	Functional and technical design is defined and can be used for coding and testing
Practices	Establishment of design documents, which are refined into lower levels that can be coded; Consideration of modularity or reusability aspects; Assurance of (internal/external) product quality characteristics; Define testing requirements; Continuous evaluation of design documents according to altering requirements; Perform updates, adjustment of design documents and ensure consistency

Process	**Development Control**
Purpose	Development of the software product according to design documents
Outcome	Development of source code and documentation completed
Practices	Development of source code according to development approach, methods, standards, Keeping of internal and external quality goals; Ensure documentation of source code and product documentation; Perform unit testing and trigger Verification & Validation processes; Submit “controlled”/ reviewed source code to repository

Process	**Continuous Code Quality Control**
Purpose	Continuously control the code quality
Outcome	Development standards for coding or documentation are kept; Quality standard of source code is achieved
Practices	Establish control and code release process before submit; Ensure versioning and write access to repository; Perform code reviews (quality control) by professional developers; Perform corrective actions if code does not adhere to quality standards; Evaluate risks, side effects and adjust test planning accordingly; Trigger peer reviews or inspections; Rework or reject inappropriate source code; Report findings and trigger SQA processes

7.5.4.2 Reviews and Inspection

“Review and Inspection” constitutes the core processes for reviewing open source code to identify defects. Simple code reviews as well as peer reviews are proposed to improve the code quality.

Process	**Code Reviews and Inspections**
Purpose	Identify defects, design gaps through reviews or inspections
Outcome	Code quality improved and defects identified
Practices	Conduct code reviews; Perform walkthroughs or code readings; Perform corrective actions to adhere to standards; Record identified defects and trigger defect handling

Process	**Peer Reviews**
Purpose	Identify and remove defects arising from peer reviews
Outcome	Improved code quality
Practices	Plan peer reviews and establish the process; Conduct peer reviews and record data; Perform corrective actions and trigger defect handling

7.5.4.3 Verification and Validation

The proposed process group "Verification and Validation" describes the management and organisation of testing and defect handling activities. Effective defect management as well as stepwise software testing with an acceptance strategy is of major importance. Software testing ought to comprise unit, integration, acceptance and regression testing before product release. As testing is performed usually within the community, emphasis on test coordination and organisation is necessary. Both, testing and defect handling processes require special importance regarding manageability, efficiency and understandability to take advantage of user testing. A continuous review of process efficiency is recommended.

Process	**Defect Management**
Purpose	Management and control of defects along the entire lifecycle
Outcome	Defects are identified, clustered, managed and controlled
Practices	Establishment of central defect reporting/tracking regarding requirements, design, source code, documentation; Ensure defect classification and prioritisation as well as risk identification; Address bug fixing tasks to community; Control and review bug fixing results; Trigger quality control before code commit

Process	**Unit, Integration and Regression Testing**
Purpose	Identification of defects and verification of the product against the requirements
Outcome	Conduct appropriate testing to ensure that the product meets the requirements; Defects are identified and corrective actions are triggered
Practices	Define test strategy; Suggest a test plan and appropriate test data; Reflect varying product configuration and system architecture; Define and document appropriate test environment; Ensure effective test management and coordination; Allocate development resources for fast defect removal time; Establish mandatory unit testing by each developer; Perform code reviews and peer review processes; Ensure the establishment of defect handling processes; Enable efficient user testing; Plan and execute integration testing; Plan and execute regression testing to ensure conformity of new features with existing environment; Report and manage defect handling; Perform re-testing in case of defects

7.6 Proposed Measurement Model

7.6.1 Objectives

The proposed measurement model focuses on the assessment of process and product quality targets as well as on project success.

The first element, the process assessment model, consists of a “process capability model and a process capability determination method” (Wang and King, 2000, p.54). The process capability model consists of scales for a quantitative evaluation of the organisation’s capability, state Wang and King (2000). The process assessment reflects the degree of excellence of an organisation, indicating their capability to produce high quality products. The capability of the organisation is assessed according to the level of distinctive processes, which implies certain quality assurance measures. The capability determination method describes the measurement approach of the process model.

The second element is the product quality assessment approach, which requires the definition of quality characteristics and measures. The product assessment is seen as a discrete element within the QA framework. Quality standards, such as ISO 9126 can be independently applied and tailored according to projects needs. However, a strong link between the process and product assessment models exists. The process model is regarded as originator and triggers the establishment, management and assessment of product quality metrics.

The third element represents the project success measurement. It focuses on an overall assessment of project characteristics, such as system and information quality, usability or user satisfaction. This independent element enables a comparison and evaluation of the measurement results.

7.6.2 Process Capability Determination

The process capability determination assesses the level of process implementation within a project organisation. The process capability scope is defined by Wang and King (2000, p.57) as "an aggregation of all the performing ratings, such as existence, adequacy, and effectiveness, of the practice which belong to the process". The method focuses on the assessment of actual processes against the process capability model and determines a capability scope for each process. Thus, the capability model delivers the nearest mapping of the framework to the applied processes. The result is a process-wise capability scope, aggregated by process domains and entire project level. Here the assessment method needs to focus on simplicity, in order to find a broad applicability in the domain of OSSD. Hence, assessment approaches, such as CMM, SPICE or ISO 15504 are far too complex and too extensive. A simple scoring model, following the ideas of Open Source Maturity Assessment methods, like BRR, OSMM or QSOS is developed to assess the process capability scope. The process capability scope is mainly determined by the assessment of product quality characteristics. The assumption is made that processes produce products as output, while "products may also be fed to processes as input" (Satpathy *et al.*, 2000, p, 97). Therefore, product quality characteristics, such as functionality, usability, reliability and efficiency are applied to measure the process itself. Satpathy *et al.* (2000) suggest a range of process criteria, which are adopted as follows:

- **Functionality** is the most important criterion to assess the process compliance, its completeness and suitability in the project environment. The assessment focuses on the compliance to standards, the completeness and the correctness of transformation from input to output.
- **Usability** focuses mainly on the understandability, learnability and operability aspects of a process. It describes the efforts to understand and to learn the process as well as the capability of the process executor to use it.
- **Reliability** regards the process fault tolerance and its maturity. It measures how reliable the process itself is and determines the failures during the process as its recoverability. The maturity is determined by the capability of the process to avoid failures because of faults in the process.

- **Efficiency** is an important criterion as it rates the process performance in terms of time behaviour or the resource utilisation as degree of output to input, like the required processing time, the resource usage or the degree of complexity compared to the process results.

The proposed process capability determination method (see figure 80) uses a scoring model to measure the degree to which level the QAfOSS processes are implemented. This method integrates two steps. First, it measures the degree of process implementation using a fit-gap analysis to identify the applied process groups and subsequent processes. Second, it assesses the fulfilment of process characteristics such as functionality, usability, reliability and efficiency. Each characteristic results in a score that increases or decreases the process score. Under-fulfilment is penalised with negative scores, while over-fulfilment is rewarded. Thus, established processes with poor fulfilment may be downgraded to zero, while negative values accordingly do not exist. The characteristics themselves are weighted, as functionality obtains 50%, usability and reliability 12.5% and efficiency 25% of the weighting scores. Functionality is the most important criterion, as accuracy, suitability and compliance of the process are regarded to be of major importance. The process results compared to the input have a high value for the project. Therefore efficiency, obtains a higher ranking as handling issues (usability) or fault tolerance (reliability).

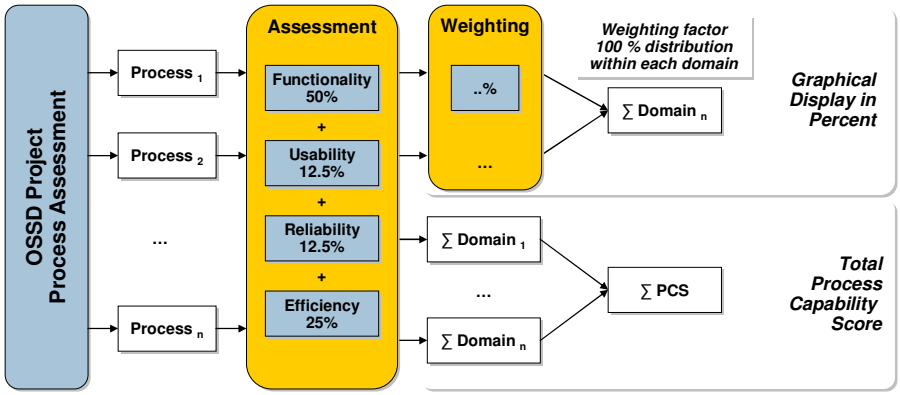

Figure 80. Proposed Process Capability Determination Method

The scoring model assesses processes in capability scores (PCS)

1. Processes availability is assessed by:
   - Existence 8 PCS
   - Non-existence 0 PCS
2. Process characteristics are assessed by
   - Functionality (+4 till -4) PCS
   - Usability (+1 till -1) PCS
   - Reliability (+1 till -1) PCS
   - Efficiency (+2 till -2) PCS

The assessment of each process characteristic results in an individual process capability score based on the degree of fulfilment, as stated in table 13.

Table 13. Proposed Assessment Criteria

Capability Score	**Characteristic fulfilled**	**Functionality** *Fu()*	**Usability** *Us()*	**Reliability** *Re()*	**Efficiency** *Ef()*
Value	Over	+4			+2
	Strongly	+2	+1	+1	+1
	Mainly	0	0	0	0
	Partially	- 2	- 1	- 1	- 1
	Not	- 4			- 2

The functions *Fu(i), Us(i), Re(i)* and *Ef(i)* determine the process characteristics. Assuming a project consists of *n* processes, the total process capability score is determined by the sum of process characteristics as shown in the following equation:

$$PCS = \sum_{i=1}^{n} (8 + Fu(i) + Us(i) + \mathrm{Re}(i) + Ef(i))$$

A total score of 368 PCS can be obtained, while the average is 184 PCS, as shown in figure 81.

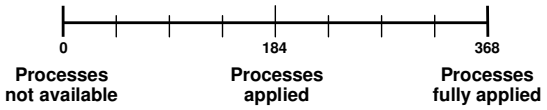

Figure 81. Overall Process Capability Scale

The process results are graphically displayed using a radar chart (or spider chart) as depicted in figure 82. This allows an overall performance representation and supports an easy comparison between different ratings. The two-dimensional graphic shows a percentage value of each domain on axes starting from the centre. The chart graphically displays the assessed capability of the rated processes with concentration on their strengths and weaknesses, normalised on a scale from zero to hundred. Thus, the chart enables an independent presentation of processes, clustered according to process objectives.

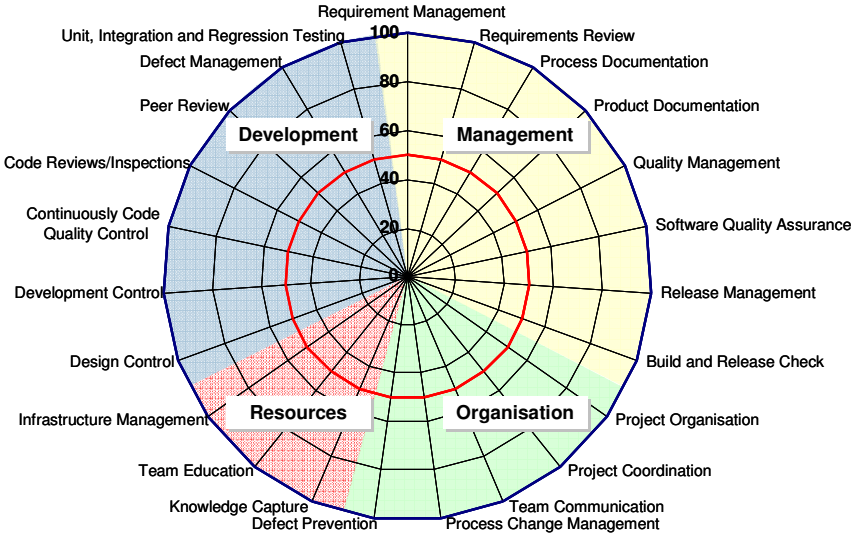

Figure 82. Process Capability Graphic

7.6.3 Product Quality Measurement

The importance of product quality measurement was discussed in chapter 2.5. The ISO 9126 consolidates the views of authors such as Boehm *et al.* or McCall *et al.* and offers a product quality measurement standard. It consists of internal, external and "quality in use" criteria grouped by the following categories: functionality, reliability, usability, efficiency, maintainability and portability. In the following research, the ISO 9126 standard is adapted to measure product quality under the OSSD. It comprises a multitude of measures and should be tailored to the project needs, because quality goals are product dependent. While some measures, such as modularity or correctness are common, projects require the determination of specific measures. For example, an interface development project may focus mainly on reliability; however, the development of a graphical user interface may regard usability as a primary goal. The definition of product specific quality goals are mandatory and can be approached using the GQM method.

The ISO 9126 is considered as an appropriate product quality measurement standard under the OSSD. However, further attention to the OSSD characteristics is necessary. The design of OSS products is constantly altering, because requirements evolve within the lifecycle. This makes it difficult to verify and validate the product. The completeness of design documents, the specification effectiveness or the design verification effectiveness are seen as important criteria, which gives significance to the category "suitability".

Code quality is a major aspect in the OSSD cycle and it is reflected by the ISO 9126 standard regarding understandability, completeness, conciseness, portability, consistency, maintainability, testability, usability, reliability, structure or efficiency.

The code accessibility (changeability) is a precondition for user testing, inspections and peer reviews throughout the entire lifecycle. It describes the measurement of the source code accessibility and requires special attention. Stamelos *et al.* (2002) have analysed the code modularity and show that high modularity leads to lower defect density in OSS projects. It presupposes a unitised specification and increases the reusability of software components. Therefore, code modularity would affect "maintainability" of "interoperability". Reusability is an important characteristic as projects make use of existing source code to reduce effort and increase their productivity. It places a high demand on software quality regarding integration aspects. Moreover, reused code could offer a high software quality, if the product is mature and comprehensively tested. In consequence, quality criteria, such as "changeability" and "replaceability" need special consideration. Furthermore, testing places a certain demand on "analysability" of the user testing effectiveness, the capability and the defect reporting quality.

In this research, no further analysis of the product quality measurement model is conducted. The exploration of an extended OSSD product measurement model may be subject to future studies.

7.6.4 Project Success Measurement

Project success measurement concerns the assessment of Information System success, which has been the subject of considerable research (DeLone and McLean 1992, 2002, 2003; Iivari 2005, Seddon 1997, Seddon *et al.*, 1999). Crowston *et al.* (2006) studied the information system success in free and open source software development and revised the model of DeLone and McLean. The DeLone and McLean (1992, 2002 and 2003) model suggests a measurement of system quality, information quality, usage, user satisfaction, individual impact and organisational impact as depicted in figure 83.

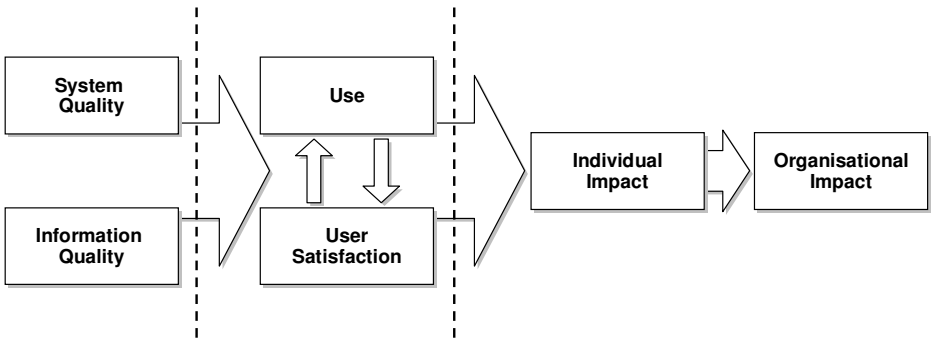

Figure 83. Information System Success Model (DeLone and McLean 1992)

The success measurement categories "individual and organisational impacts" focus on economic and other influences. Individual and economic impacts do not primarily reflect quality

criteria. Therefore, these categories are disregarded in the project success measurement model. The categories "system" and "information quality", "use" and "user satisfaction" reflect the primary quality target. These criteria are applied to obtain an alternative view of software quality expressed in terms of project success criteria.

The project success measures of Crowston *et al.* (2006) are shown in table 14. Their research delivers a theoretical informed range of measures that are revised in an empirical study of OSS projects. As a result, OSS success measures, which focus on system creation and maintenance, system quality, system use and system consequences have been developed. In this thesis, the study of Crowston *et al.* (2006) is considered as the starting point for the OSS project success measurement. Moreover, the IS success measurement approach is used to assess the multifaceted concept of software quality. The relation between IS success measurement and software quality are shown. First, the model measures the system and information quality, which comprises code and documentation quality. Thus, it reflects the manufacturing view of quality. Second, the user satisfaction measures the rating of users or their opinion, while the category "use" focuses on the usability characteristics of the product, expressing its popularity. These criteria reflect the user's view of software quality and describe how the product satisfies requirements. This view is emphasised even more with the high popularity of a product.

Table 14. IS Success Measures in the context of OSS

Process Phase	Measure
System creation and maintenance	Activity/Effort Attraction and retention of developers (Developer satisfaction) Advancement of project status Task completion Programmer productivity Development of stable processes and their adoption
System quality	Code quality Manageability Documentation quality
System use	User Satisfaction Number of Users Interest Support effectiveness
System consequences	Economic implications Knowledge Creation Learning by developers Future income and opportunities for participants Removal of competitors

The aim of the success measurement approach is the development of an independent project success assessment in the OSSD domain. The quantitative assessment is based on metrics,

which require the definition of a metrics model and a project success determination method. The metrics model follows the research results from Crowston *et al.* (2006) and is restricted to product and process quality and usability. The project success measurement model (as shown in table 15) adopts the categories "system creation" and "maintenance", "system quality" and "system use", while "system consequences" are neglected due to their economic perspectives. The complete metrics model is shown in Appendix B.3.

Table 15. OSS Project Success Measures

Process Phase	Measure	Metrics
System creation and maintenance	Activity/Effort	Number of source code changes, Number of developer mailings, Activity of issue tracker
	Attraction and retention of developers	Project Size, Developer team size, Developer community growth, Developer regeneration / fluctuation, Continuity
	Advancement of project status	Release status, Maturity, Vitality, Market availability
	Task completion	Release frequency (average), Release activity, Time to fix bugs, Short feedback loops
	Programmer productivity	LOC per developer, Developer community heterogeneity, Effective leverage of user community
	Development of stable processes and their adoption	Process documentation, Developer documentation, Process automation
	Testing effectiveness	LOC tested per user, Average defects found per user, Efficiency of user testing
	Source code access	Source code accessibility
System quality	Code quality	Modularity, Correctness, Coupling, Complexity
	Manageability	On-boarding time, Amount of abandoned code
	Documentation quality	Source code comment, Outdated code documentation, Inadequate code documentation, User documentation up-to-dateness
System use	User Satisfaction	User satisfaction checking
	Number of Users	Community size, Downloads, Mailing interests
	Interest	Popularity
	Support effectiveness	Questions answered, Implemented requests, Support effectiveness

The proposed project success determination method describes a metrics-based scoring approach as depicted in figure 84. Metrics deliver numerical values according to a measuring unit. As the value itself does not reflect any valuation regarding information system success, a clustering or a respective scale is introduced for each metric. For instance, the value of a determined bug fixing frequency is ranked into different success categories. These results

enable a comparison of different projects within a selected metric, but make an evaluation between metrics impossible. The addition of measured results to a total score does not deliver any useful results unless a common scale is used. Therefore, all metrics are normalised to a project success scale ranked from one to five. “Five” represents fulfilment and “one” non-fulfilment. The results are summarised by categories to determine a total Project Success Score (PSS). A percentage-weighting factor with one hundred percent distribution across all metrics is used to emphasise certain quality factors. The initial weighting is predefined according to the researcher’s assumptions based on observations in the literature and survey research. The initial weighting factor can be found in the Appendix B.5 (table b.3) and is reviewed within the further research.

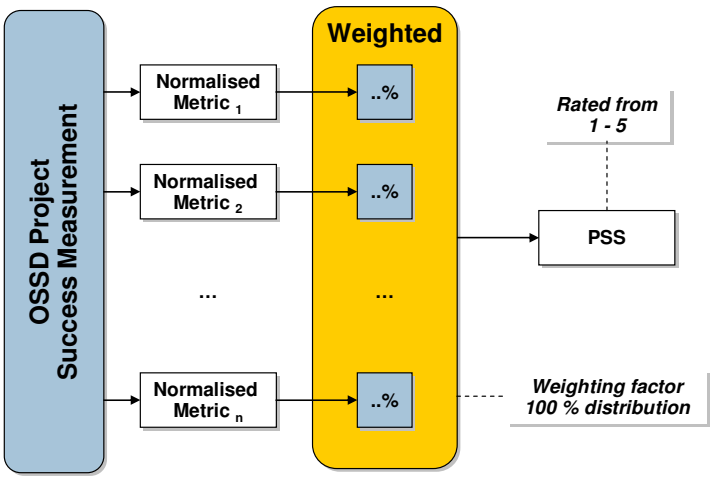

Figure 84. Proposed Project Success Determination Method

Each metrics M_i is normalised to a scale from [1 – 5] and weighted with w_i, where w_i is in the range of [1 – 5]. Assuming, the measurement model consists of n metrics, the Project Success Score (PSS) equation is as follows:

$$PSS = \sum_{i=1}^{n} \frac{(M_i * w_i)}{\sum_{i=1}^{n} w_i}$$

A threshold of 3.0 defines the separation between a project disappointment or success, as shown in figure 85.

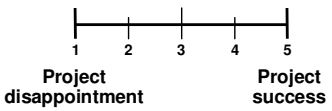

Figure 85. Project Success Scale

7.7 Framework Consolidation

The proposed Quality Assurance Framework for Open Source Software describes an overall approach to improve and determine software quality assurance processes. It follows the idea of the SQP (Baker and Fisher 1999) and combines a process model, an assessment model and a determination method. These elements are consolidated in the proposed framework showing their application, impacts and dependencies, as depicted in figure 86.

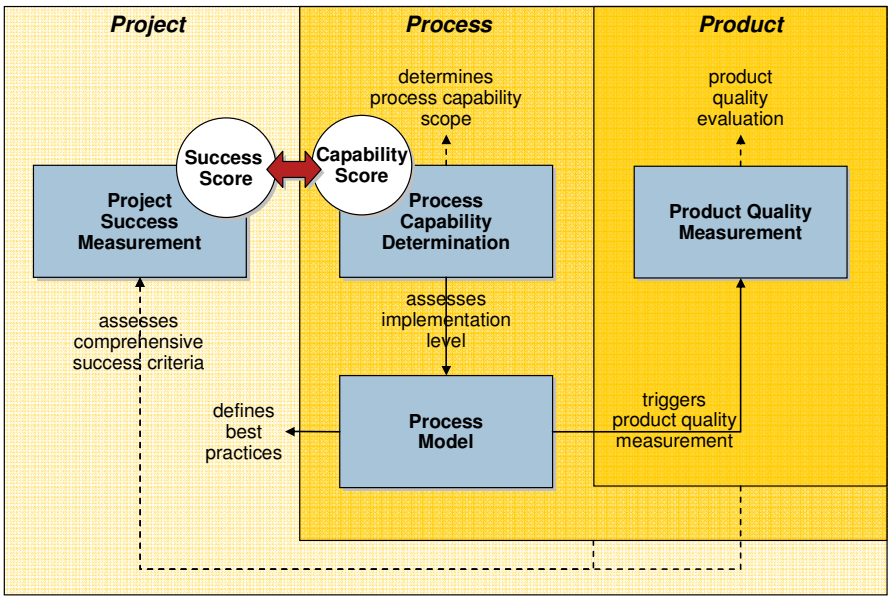

Figure 86. Proposed Consolidated QA Framework

The core element is the process model that provides a set of processes, focusing directly and indirectly on software quality assurance. Projects can use the process reference model to review or improve their procedures. The model is easily tailorable to specific needs through an extension of additional characteristics or through deselecting unusable processes. The process model requires the establishment of individual product quality measurement using metrics. The ISO 9126 standard is adapted using the GQM method to projects' specific requirements.

The process capability determination and the project success measurement model represent the second pillar of the QA framework. The process capability determination method provides a capability scope that reflects the degree an organisation fits the process model. This degree is quantified in form of a capability score and expresses the QA process capability of an organisation according to best practice.

The second element is the project success measurement approach. It represents an autonomous assessment to evaluate success in Information Systems (IS). Furthermore, IS success is used as an indicator for software quality. Projects considered as successful, if the product satisfies the user's needs, the project has a satisfactory development approach and it attracts

participants. The project success measurement method is used as an independent control element for software product and process quality. It delivers quantified results and enables the benchmarking of projects.

7.7.1 Framework Assumptions

The correlation of process capability scope and project success is the subject of further analysis. The general assumption is that projects with a high process capability scope will achieve a high project success score. A high project success score could indicate high software quality (for the reasons discussed in section 7.6.3). High process capability scope shows that the model fits the applied processes. If a process model fits the processes that appear to deliver high quality, the implementation of them could lead to improved software quality assurance. This is based on the assumption that the process model is comprehensive and has a high degree of coverage within the analysed processes. Otherwise, the causal relationship between the model implementation and the SQA improvements is misleading. Hence, if a positive correlation between both measures PCS and PSS exists, the implementation of the QA framework contributes to the improvement of software quality assurance.

Thus, the following assumptions are made:

- There is a high degree of coverage of the process model within applied processes in mature OSS projects
- The project success measures are an appropriate indicator for process maturity and software quality
- A fully implemented QA framework results in a high process capability score
- A correlation between the process capability and the project success score exists
- A high process capability and a high project success score show that the QA framework could contribute to an improved software quality assurance

These assumptions are examined using an ideographic method with case studies. A validation of the model with actual results provides evidence for the hypotheses.

7.7.2 Framework Implementation

The implementation of the QA framework follows an iterative approach, based on the Plan-Do-Check-Act cycle after Deming (1988). This is similar to the OSSD model, which is a recurring development cycle with continuously altering targets. OSSD projects evolve during development, while processes mature and goals alter. This demands the implementation of a continuous process improvement that assists projects in their entire development lifecycle. As opposed to Deming, the initial start of the PDCA cycle is the “initialisation” phase whose

primary goal is to verify the actual processes against the reference model. The proposed improvement process cycle is depicted in figure 87 and implemented as follows:

- The definition of the project goals and the quality goals is mandatory
- The reference model is selected and tailored to the project needs accordingly
- The applied QA processes are assessed and deliver the actual process capability
- Based on the analysed results further process improvements are planned
- These improvements must consider varying needs and evolving targets
- The cycle iterates with the implementation of suggested measures

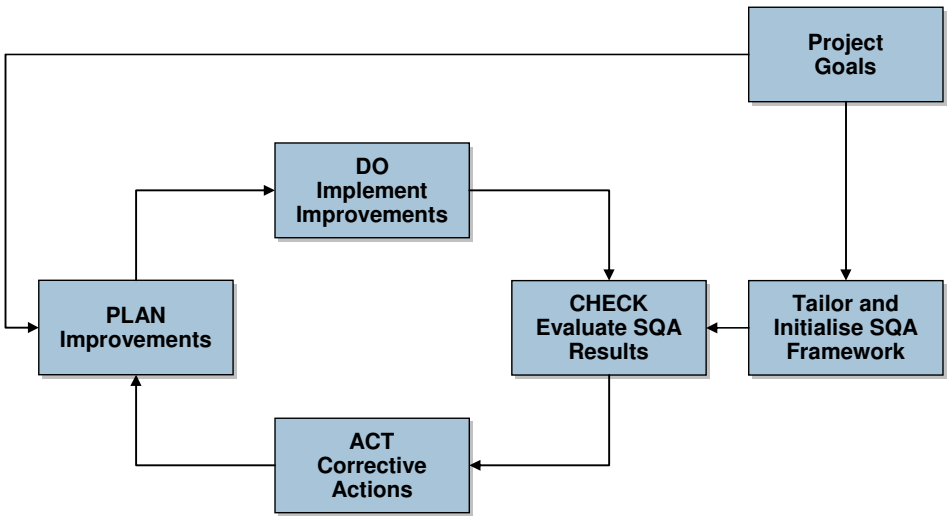

Figure 87. Framework Implementation

7.8 Summary

The development of the quality assurance framework proposed in this chapter follows a stepwise research approach suggesting a chain of activities, such as the definition of the model, the specification of the model components, the definition of measures and the collation of the model. The stakeholder interests were discussed and the user's and developer's view were obtained as directions for the framework. The characteristics of the OSSD model and the factors that influence software quality, such as documentation management, peer reviews and inspections or intensive user testing were highlighted. Furthermore, the necessity of QA assurance processes was analysed in order to define common process groups with subsequent processes. A process-oriented model was considered as the foundation of the framework. Therefore, a common process structure and a generic process framework are proposed as the underlying structure. Quality models, such as CMM, ISO 12207 or assessment methods, such as SPICE were analysed and the findings adopted for the framework

development. The proposed process reference model represents the core of the model. It triggers the implementation of product quality measures as suggested in the ISO 9126 standard. An assessment model was proposed, comprising a process capability determination method and a project success measurement method. In order to complete the QA framework these elements were consolidated and their interaction was shown. The QA framework fulfils two goals. First, the reference model supports projects enhancing their quality assurance processes. The process capability determination assesses the degree of process realisation and assists projects to improve their processes. Therefore, a continuous improvement approach, following Deming's PDCA cycle, is suggested for the framework implementation. Second, a comparison of both assessment approaches PCS and PSS provides further evidence for the validity of the model. The correlation of process capability and project success measures is explored in the following case studies.

8 Validating the Proposed QA Framework

This chapter describes the case study research adopted to validate the proposed QA framework. The underlying assumptions and the scope of the case studies are defined. The units of measure are analysed and the site selection is explored. The suggested QAfOSS framework model is applied in the case studies and the findings regarding process and project success measures are discussed. A statistical analysis is conducted to verify the model. The chapter closes with a discussion of the findings.

8.1 Assumptions and Research Direction

8.1.1 Research Question

The second research question (as defined in section 1.1) implies the establishment of the QA framework and the examination of its contribution to SQA, as stated in section 7.1. In the case studies, the applicability of the QAfOSS framework and its potential for improvement for the OSSD is examined. Thus, this section focuses on the main intention of the research question:

> *Can a quality assurance framework contribute to the improvement of the quality assurance activities in the OSSD?*

This leads to subsequent questions:

- What is the degree to which the QA framework correlates with applied practices?
- How do the project success measurements represent an appropriate basis for comparisons?
- How do the QA framework processes correlate with the project success metrics?
- What improvement of QA practices could be derived from the observations?
- What limitations does the approach have?

8.1.2 Presupposition

The case study experiments are based on the QAfOSS framework and apply the proposed process and success measurement approaches. The selection of different projects, regarding their type, development approach and level of success is considered for validation purposes. The measurement model is used to evaluate applied processes and indicates the degree, to which the project matches the framework. The premise for this experiment is that the measurement model delivers comparable results any time in the lifecycle. Hence, an evaluation and comparison of the results against their success and quality characteristics, shows a comparable result regardless of whether the QAfOSS model is introduced directly from the start or assessed at later stages in the lifecycle. A high process and success level express compli-

ance with the model and show its validity. Thus, the assessment approach is considered as appropriate for validation, as opposed to testing the model in longitudinal studies.

8.1.3 Scope and Objectives

The scope and objectives of the case studies are defined as follows:

Scope:

- Conduct of a structured interview and observation of the project documents and deliverables
- Completion of the questionnaire by project members to determine process capability and project success levels
- Validation of research questions using qualitative and quantitative data analysis

Objectives:

- Identification of applied development approaches and methods
- Analysis of the QA framework processes with current practices in OSS projects
- Determination of a rough fit-gap degree to the process model
- Assessment of preliminary findings and clustering of projects according to replication criteria
- Analysis of project success determination results in correlation to the estimated and actual project success
- Analysis of correlation between process capability and project success measures

8.2 The Case Study Research Approach

The case studies focus on the exploration of the research questions and are based on developed quality assurance theories, in order to gain a uniform concept for organisations. Case studies allow an analytical generalisation of results (Yin 2002). Furthermore, generalisable concepts can be achieved using multiple case studies (Pettigrew 1985). Bonoma (1985) in Benbasat *et al.* (1987) argues that case studies help in hypothesis testing and require multiple cases to confirm the existing theory and a single critical case for disconfirmation. The conduct of multiple cases has the advantage of a cross-case analysis, which "yields more general research results" (Benbasat *et al.*, 1987, p.373).

The case studies are conducted, using multiple methods of qualitative and quantitative techniques. Lubbe (2003) argues those techniques are not conflicting but are rather complementary. The combination of multiple methods enable triangulation, which lends greater support to the research conclusions (Benbasat *et al.*, 1987, p.374). The reasoning is drawn from the qualitative information resulting from interviews, documentation or archival records, which are supported by the quantitative findings from the questionnaire. Jick (1979) assert that the various methods together produce largely consistent and convergent results. The question-

naire results become more meaningful when coordinated with critical qualitative information about underlying approach or development direction, and then findings may appear in a different light. Lubbe (2003) suggests, whenever possible, converting purely descriptive data into quantitative data to perform statistical analysis. Therefore, the analysis is performed on the quantitative findings from the questionnaire to determine the correlation between applied processes and the project success using statistical methods.

8.2.1 Unit of Analysis

The case study approach is used to describe the relationships that exist in organisations (Galliers 1985). The research approach obtains an objective interpretative position to explore and analyse different organisations, as discussed in section 4.4. The selection of the appropriate unit of analysis is of major importance. OSS organisations have varying structures with diverse social complexity that are not under the control of the researcher and may have an impact on the research results. Thus, it is not possible to find a control group that differs only in a few variables. Yin (2002) argues the case study approach is a good way to understand complex social phenomena and its strength is the ability to deal with a full variety of evidence. This approach is needed to explore different organisations and to understand how software quality is effectively assured. The units of analysis are organisations developing OSS products. Rather than "organisation" the term "project" is used, as it seldom affects legal organisations and describes more precisely the voluntary collaborative development approach. In order to obtain a larger variety of cases, the selection of projects is based on the degree of fulfilment of the OSSD model, the project complexity, the maturity level, the development approach and diversity of application.

- The requirements of the OSSD model need to be fulfilled regarding the openness of the source code, the voluntariness of participation and the geographically distributed user-driven development through web-interaction.
- Projects face a certain degree of complexity, which correlates with their size. Communication and coordination efforts scale up with increasing team size. While debugging tasks could largely benefit from parallelisation, development is less scalable and requires more communication. Larger projects need to manage an increased amount of collaboration processes and may require more precise processes or standards than smaller ones. Small projects could refine their deliverables on request and have less need for guidelines. Therefore, mid- to large sized projects are within the scope of this study.

- The study focuses on mature projects that achieve a productive release status and operate longer than a year in the market. It is assumed that the more advanced the project, the higher the expected process characteristics.
- Different development models are of interest using purely an OSS approach or a hybrid model, which combines e.g. a commercial traditional development with the OSS approach.
- A high degree of coverage can be obtained if the varying interests and requirements of the diverse applications are considered. Therefore, different established OSS projects of the categories ERP/CRM/Financial, Office/Business, Internet and Software Development are chosen. This enables the identification of influencing factors between projects.

In summary, the unit of analysis concentrates on mid to large size OSS projects of mature status, with at least one year in the market, which have more than five active developers and a community of more then fifty users. The case studies are conducted with members of projects who play a leading role such as core developer or project manager.

8.2.2 Site Selection

Yin (2002) proposes the consideration of two criteria to determine the site for case study research; the literal replication is used where similar results may be predicted while the theoretical replication is used where contradictory results may occur. The selection of an appropriate site requires more insights into the object of study. A structured interview is used to collect data about applied processes, development approach and methodology. The findings of the interviews are used to classify organisations according to a literal or theoretical replication. The case study was conducted only with selected organisations to ensure the required diversity. Therefore, different unit types of analysis and types of replication, as shown in table 16, are selected.

Table 16. Site Selection Criteria

Criteria	Type of Application	OSSD Model	Maturity	Complexity	Development Approach	Type of Replication
Case 1	Internet	Fulfilled	High	Middle	Commercial driven OSS	literal
Case 2	ERP/ CRM/ Financial	Fulfilled	High	Low	Community driven OSS	literal
Case 3	Internet	Fulfilled	High	Middle	Commercial driven OSS	literal
Case 4	Software Development	Fulfilled	High	Large	Community driven OSS	literal
Case 5	Office/Business	Fulfilled	High	Large	Commercial driven OSS / Hybrid	literal
Case 6	Office/Business	Partially Fulfilled	High	Large	Commercial driven OSS / Hybrid	literal
Case 7	Software Development	Fulfilled	Low	Middle	Community driven OSS	theoretical

8.2.3 Data Collection Approach

The richness of the data, as well as capturing the contextual complexity, can be achieved using multiple sources of evidence (Benbasat *et al.*, 1987). The data collection method comprises the following sources:

- Structured interviews with open-ended questions
- Documentation and archival records
- An evaluation questionnaire with closed-ended questions and ordered scales.

In order to obtain a further insight of the development approach, structured interviews were conducted with leading developers or project managers in the selected organisations. A questionnaire was used as guideline, the outline of which can be found in the Appendix B.1. The guideline has a unique structure to allow a better comparison between the projects. However, adherence to the guideline is flexible to ensure the capture of as much additional information as possible. All information was recorded on a data sheet.

Documentation and archival records came from various sources, such as the project content in the web portal, the code repository or discussions in mailing lists. The qualitative information was captured in a textual form.

The evaluation questionnaire was used to record detailed information about applied quality assurance processes and project success measures, as outlined in the Appendix (B.2 and B.3). It describes the QAfOSS framework that combines the process model and project success measurement approach with its proper determination method. The development of the questionnaire followed the survey instrument design method. Content validity is established through reviews, as suggested by Straub *et al.* (2004). A pre-test was conducted to verify reliability and construct validity. All quantitative data was captured in a spreadsheet for statistical analysis.

8.3 Case Study Findings: Process View

The case study results are presented anonymously for the reasons stated in the ethical discussion in section 4.2.3. The selected projects are classified according to their characteristics and the application type. Each case study comprises data from interviews, the researcher's observations and the process evaluation results. A qualitative analysis of the findings is more important than a quantitative measurement as respondents applied different evaluation standards. Therefore, the quantitative analysis is used only to support the findings and to show statistical correlations. In a critical discussion, major tendencies, strengths or weaknesses of the projects in relation to the QAfOSS model are argued.

8.3.1 Case Study One

The first case analyses an Internet application development with medium complexity. The commercially driven OSS project consists of a small employed core developer team with a mid-sized community of 50-100 persons. The main drivers for this project are commercial interests, which impacts leadership and motivation and drives constant improvement of features. The project organisation is informal, no hierarchies exist. The main communication is via a mailing list and a Subversion (SVN) activity list is used to track the changes. Requirements are captured from various sources, such as list of required features and are discussed in a developer conference. The documentation of processes mainly focuses on style and coding guidelines, as complete user and technical documentation exists. The quality of the user documentation is relatively poor as its up-to-dateness is behind the technical documentation. Knowledge transfer processes do not play an important role, as an in-house core team coaches new developers. The mailing list or the version control tool is used for knowledge capturing and for understanding the previous development. The development processes are largely supported by tools. An intensive design phase was conducted to establish a solid architecture and a well-defined class concept. Continuous quality checks are performed to control the code quality. Furthermore, unit tests are mandatory before code is committed to the repository. The core team informally performs inspections or peer reviews. Defect handling and testing processes are well structured, as a full testing suite, with automated tests is applied. Release management is mostly feature-based, while a time-based approach is used when it comes closer to the next release date. The project focuses on product quality features via performance profiles or benchmarking, but neglects the management of quality targets. SQA tasks are indirectly executed and mostly performed by the project management. Continuous processes improvement activities, such as defect prevention or process change management are not applied. In summary, a good compliance with the process model exists, but there are major deviations in the quality management, improvement and knowledge transfer processes, as shown in figure 88.

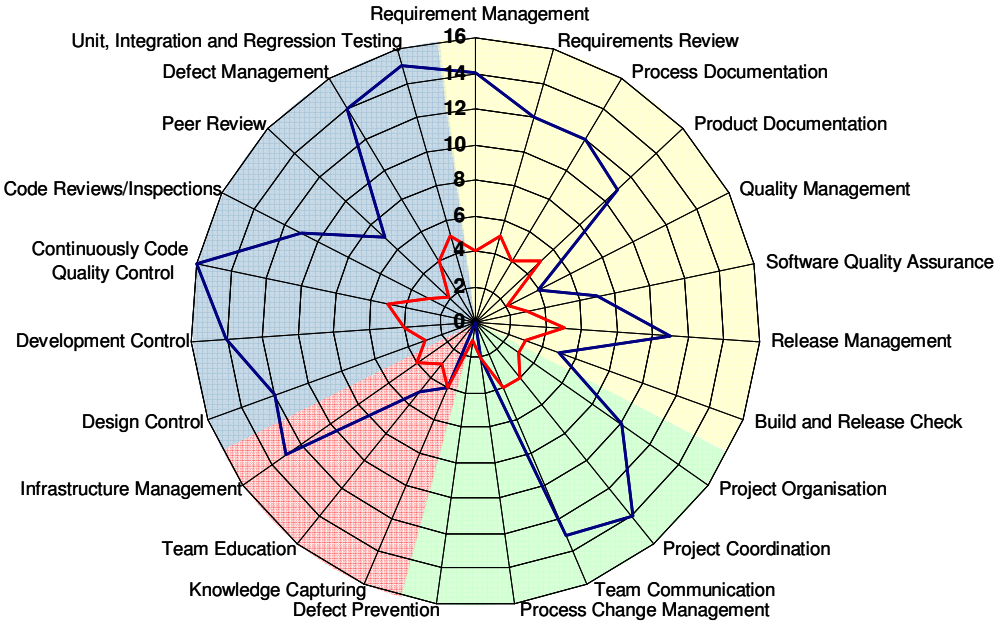

Figure 88. Analysis of Case Study One

Note: The blue graph represents the obtained process score, while the red graph shows the weighting factor for each category.

8.3.2 Case Study Two

The second case focuses on an ERP/CRM/Financial application of medium complexity. The project is driven by personal and company needs, and constitutes a purely OSS based model. The community defines the development direction. The project is loosely organised, while some core team members perform project management tasks. Due to the small team size of around 10 developers, there are fewer needs for distinctive processes. Sometimes processes are handled informally. The project management is less strict but focuses on the coordination and integration of project tasks. The project shows similar tendencies to Case 1, but has a more pronounced process compliance with the QA framework. Reviews, inspections or peer reviews are largely applied. SQA processes are completely informal, which devalues this aspect, as shown in figure 89. Team education, knowledge capturing and defect prevention processes are informally applied, which leads to a slightly lower rating. On average, there is a high compliance with the QA framework.

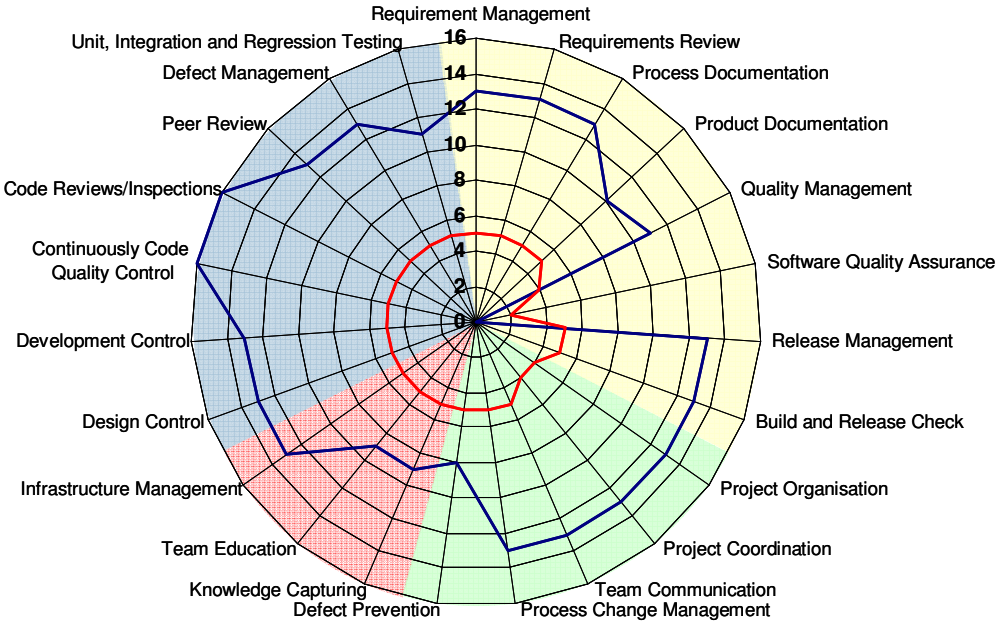

Figure 89. Analysis of Case Study Two

8.3.3 Case Study Three

The third case explores a commercially driven Internet application of low complexity. The project applies the Apache project standards and is ambitious with a high commercial motivation. The development processes are well organised, a highly professional and knowledgeable team exists. Requirement management processes are not centrally organised, as the project WIKI, bug lists or the issue tracker are used to collect feature requests. Both process and product are well documented. The technical documentation comprises build information and contains examples, while training documentation is being planned. Project organisation and communication follow the rules of the Apache foundation. All development processes are largely supported by tools. The project WIKI is used for documentation and knowledge capturing. Any educational perspectives are not relevant, as the contribution of professional developers is preferred. The design of the application is documented in the WIKI, but a development control does not exist. Developers are encouraged to perform their tasks correctly and need to perform mandatory unit tests before commit. Any changes are published in the mailing list, to enable continuous reviews by the community. Review and inspection processes are handled informally, while peer reviews are seldom undertaken. Verification and validation processes are well structured. A defined testing strategy exists with a set of defined test cases. Testing comprises unit and integration tests, as well as automated tests. Release management follows the Apache guidelines, using release candidates for further verification. Quality management focuses mainly on code coverage analysis in order to verify the code quality. SQA processes are implicitly applied following the Apache standards. The

projects neglect continuous improvement processes, as process changes are informally handled and only the philosophy of a defect prevention approach exists. In summary, the project shows a high process compliance and a good coverage of the QAfOSS processes, as depicted in figure 90.

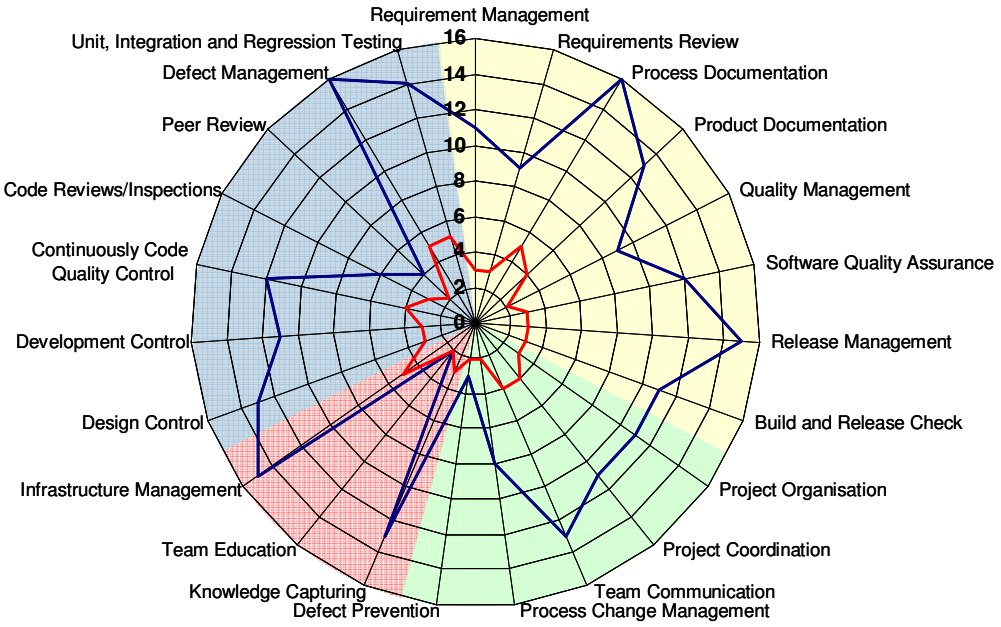

Figure 90. Analysis of Case Study Three

8.3.4 Case Study Four

Case 4 explores a very large software development project of high complexity. The projects acts and operates like a large traditional development project, but following the OSSD model. A strong leadership drives the project direction. Several commercial interests exist, as paid developers contribute to the project. Requirements are collected from developers, user mailings, discussions in mailing lists and commercial partners. They are centrally documented in a WIKI. The core development team or the engineering team continuously performs requirement reviews and updates, as new features require an approval by the core team. The process and product documentation is largely available, development and coding style guidelines exist, best practices are documented, technical and user documentation is published on the WIKI. The project has a flat organisational structure and follows the OSSD role model. The management team ensures the project coordination and a bug tracking tool is used to track general tasks. Communication is based on public and private mailing lists and instant messaging. Although there is no direct focus on knowledge capturing, mandatory documentation with every committed code is required. For external contributors stronger quality checks are applied but all code is reviewed prior to release. The project applies distinctive team education processes, which comprise the monitoring of the developer skill lev-

els and ensures that only developers are appointed who have achieved certain ability. Furthermore, information is spread via mailing lists and training is provided or conferences are attended. Tool standards are well established and documented. The analysis of the engineering processes shows, that the system design is documented in the WIKI, but often is not up-to-date. A test suite is used to test against the new features, however the design control process is lacking. There is no clear pragmatic approach available to control the development, except some development guidelines. A continuous code quality control is undertaken. Nevertheless, a gap between documentation and code exists and a tool to check the status of the documentation is not available. All kinds of reviews occur on community interests, because no formal process exists, except the posting of changes via the mailing list. The defect handling processes are supported by Bugzilla and comprise status tracking, classification and prioritisation. Nightly builds are carried out to check the overall stability, a testing suite is used to perform automatic testing and test cases are defined by the developers. Unit tests are only partially required, depending on the part of the application and its testability. The release management changed shortly from a feature- to a time-base approach that offers better control for the project team. The engineering team verifies the stability of a release and builds release candidates. No additional testing prior or post to a release is scheduled. The development team makes heavy use of benchmarks, while the community tests. An in-house SQA team ensures the execution of quality assurance tasks. The project management has defined quality targets and goals; even benchmarking is done with strict measurement. However, the quality management is rated as “not applied” in the questionnaire. Improvement processes are informally applied in the team, which leads to the devaluation of the “process change management”. The analysis shows a high compliance with the model, as illustrated in figure 91. An even higher score would have been given, if the same process evaluation standards had been applied. For instance, the evaluation of “quality management” and “process change management” would be more highly rated after the in-depth study and the researcher’s observations.

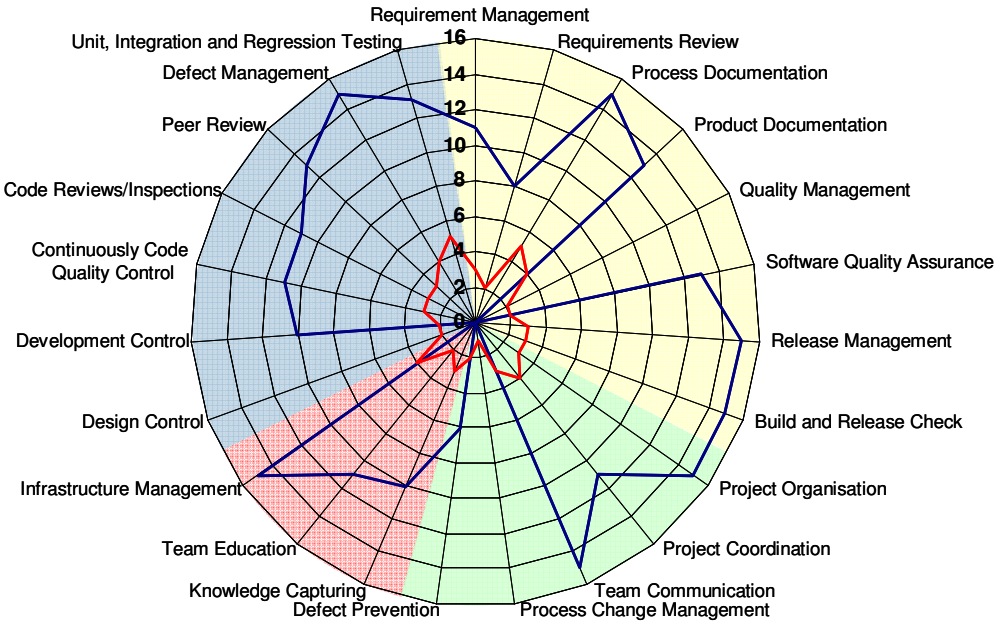

Figure 91. Analysis of Case Study Four

8.3.5 Case Study Five

The fifth case study explores a large Office/Business development project with a high complexity level. The commercially driven project follows a hybrid approach and combines an in-house development with the OSSD model. The development tasks are well structured and managed. Requirements are managed and controlled and an issue-tracking tool is applied. Formal reviews are performed by business analysts and the community before new feature requests are integrated into the design. Process documentation concentrates only on development and coding style guidelines but neglects an overall view. Both, the technical and the end-user documentation are comprehensively available. Project organisation is hierarchically organised. A well-structured issue-tracking tool is used for coordination purposes. The management team steers the project with focus on the roadmap and the scheduling of in-house resources. The communication is set-up using different channels, such as instant messaging, public/private mailing lists and user groups. However, communication is shifted mainly to user groups, as it enables easier tracking of historical information. Knowledge capturing processes are implicit done and concentrate on mandatory documentation for every piece of code. Team education includes presentations, public information, conferences or coaching by internal developers. The project has not defined any infrastructure guidelines, as a large variety of tools are required to test the readiness for every platform. The system design is informally controlled. A management commit is required and a solid architecture has been defined. Moreover, the project conducts automated tests against design documents. Beside manual checks, automated code style checkers are used to control the development activities.

The project applies pair programming techniques and has a high focus on inspections and peer reviews. For instance, the whole team continuously performs quality controls before commit that are triggered as a result of posted changes in the mailing lists. Verification and validation processes are well structured, using classification and prioritisation of issues. Unit tests are mandatory with all committed code. Furthermore, a dedicated QA team is responsible for risk based testing and the approval of new releases. It also coordinates the fixing of defects prior to a release. The release strategy is time-based and combines frequent OSS releases with commercial releases at longer intervals. Prior to new releases a feature freeze is conducted and release candidates are used for community testing. The project follows a well defined test strategy. For example, integration tests are conducted prior to a release, while in addition a complete regression test is internally performed before issuing the final commercial release. No formal quality management targets are defined, but there is performance testing and benchmarking. The project has a high focus on process change management activities and improvements. For instance, regular process reviews are conducted, risk mitigation meetings are held and the whole team is encouraged to prevent defects. In summary, the project is well established with mature processes and shows a high compliance with the QA framework (see figure 92). The project follows a commercial approach with full-time paid contributors and professional management. Thus, it uses the community efficiently and largely benefits from testing, debugging and user suggestions.

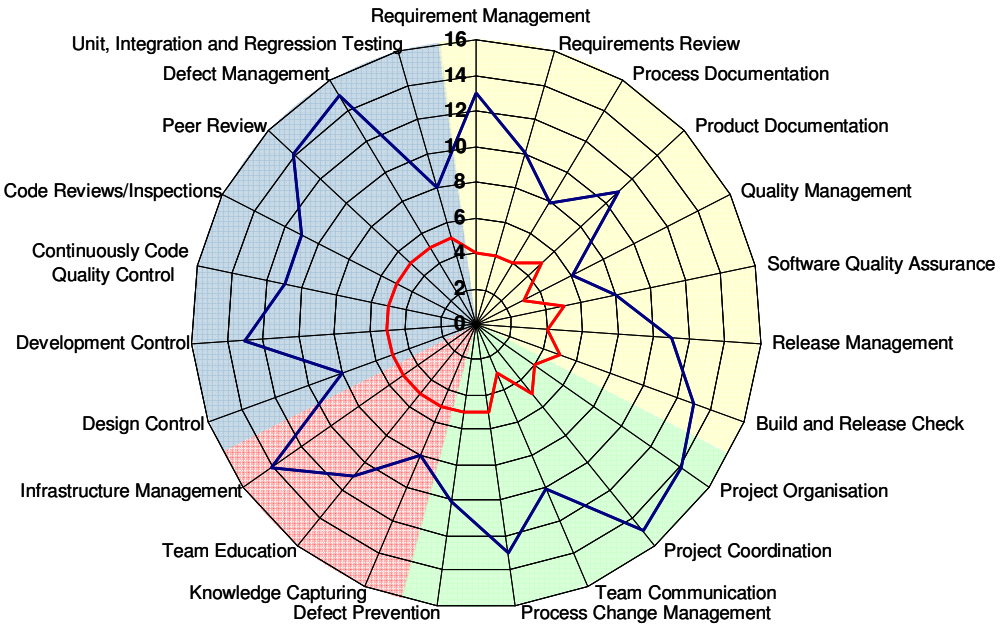

Figure 92. Analysis of Case Study Five

8.3.6 Case Study Six

This case study explores a very large Office/Business application development, driven for commercial reasons. It is managed by a company, which pays a team of full-time developers. The internal team mainly leads the development and relies on direct communication and experienced team members, while the community has minor impact on the development tasks. Thus, this project follows a hybrid approach. The predominant traditional development approach has an impact on the management, the coordination, the communication and the organisation. For instance, requirement management has different priorities for customer and community feature requests. Feature requests are discussed in the group and published on the WIKI. A voting process decides which requests to implement. Documentation of processes is internally available, while public process documentation is lacking. The project offers a large variety of technical and user product documentation. The organisational set-up is strict and access to the repository is restricted. The lead developer centrally coordinates the tasks. The project communication is via instant messaging, mailing lists or phone. More often issues are only internally handled. Knowledge transfer processes are similar to previous case studies, as code documentation is mandatory and coaching is used to integrate developers. No infrastructure guidelines are defined, because this kind of information is internally spread. Engineering processes consider design and development control, but neglect a continuous code quality control. The project relies on the community and handles reviews and inspections informally. Defect management is well structured, but testing has large deficits. For instance, no strategy is defined, mandatory parts are documented in a checklist and integration tests are only planned. The project relies on community testing prior to a new release, using release candidates. Quality targets, SQA activities as well as continuous process improvement approaches are informally managed. The project tends to open its processes increasingly to the public. The case shows a high process compliance, with some restrictions, such as testing or process change management (referring to figure 93).

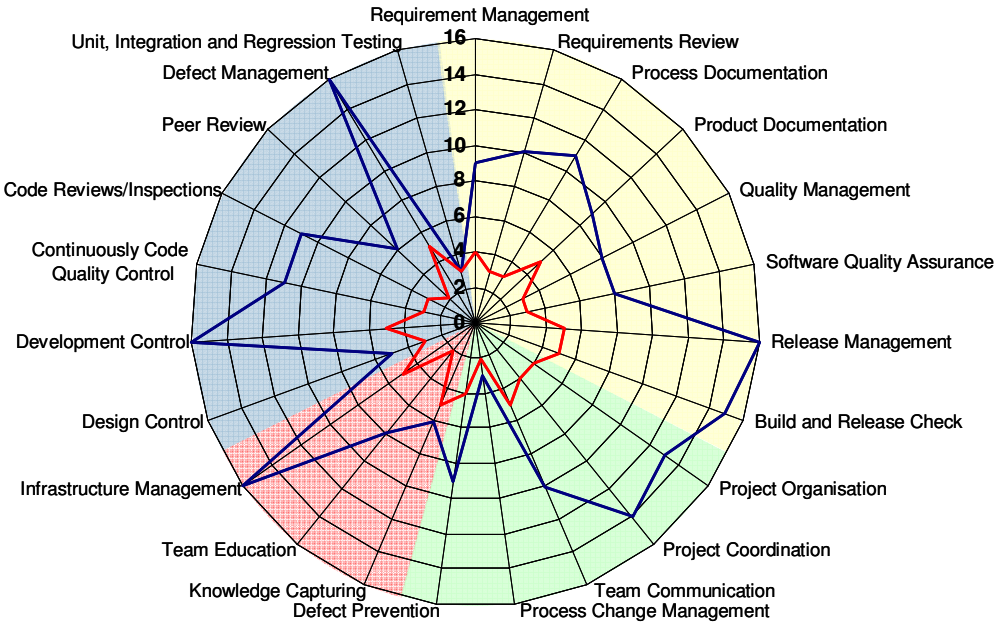

Figure 93. Analysis of Case Study Six

8.3.7 Case Study Seven

Case study 7 focuses on a theoretical replication and analysis of an OSS development project of medium complexity. Unlike the previous studies, this project has a low maturity level. It is driven by part-time volunteers and shows mid-to-low activity. Project processes are lacking due to several issues, such as organisational set-up and management. There is no strict management approach to steer the project and to ensure an appropriate environment. Requirements management processes are mostly neglected and only a simple capturing in "To-Do" lists or in the WIKI documentation exists. Transparency about required features and their status is lacking. The community performs requirement reviews informally. The documentation comprises coding style guidelines or testing guidelines that are published on the project WIKI. However, the project documentation is messy, outdated, lacking in structure and scattered. The organisational set-up follows the classical OSS role model with a flat structure and restricted access rights to the repository. The coordination of tasks occurs via the bug lists, which has a limited functionality. The main communication is via mailing lists, in order to spread and to capture information. Support for new contributors does not exist, except loosely planned coaching by experienced developers. The project uses an OSS tool based infrastructure with limited functionalities. The work environment is underdeveloped. For example, a defect handling tool with insufficient control for tracking and coordination is used. This has a negative impact on the efficiency of testing processes, because reporting quality or a timely feedback to community members is missing. Furthermore, the defect handling process is not well maintained and some defects remain for years. The quality control

of engineering tasks is informal. Nevertheless, code reviews and inspections are continuously performed, as soon as changes are posted via the mailing list. No testing strategy, planning or documentation is available and the project shifts testing activities to the community. The project follows a time-base release management approach with defined targets but an outdated schedule. Meanwhile, a large discrepancy between planning and actual release version exists and no maintenance or adjustment to the plan is made. The project follows the idea of release candidates towards the development of a stable version. Quality management targets and SQA tasks are lacking and are not documented. The philosophy for continuous process improvement exists in the community, but there is no willingness on the part of management to drive such advancements. The central issue is a lack of management skills, because the development environment is insufficiently established. There is no focus on central tasks, such as organisation, documentation and the triggering of SQA activities. Furthermore, the project is short on resources, due to part-time contributors with time constraints. In conclusion, the project cannot leverage its community efficiently and risks the loss of attraction, as infrastructure is insufficient and environmental processes are not transparent. The case study shows a low process compliance with the QAfOSS framework. Management, organisational and resource related activities especially are beyond the process targets, as depicted in figure 94.

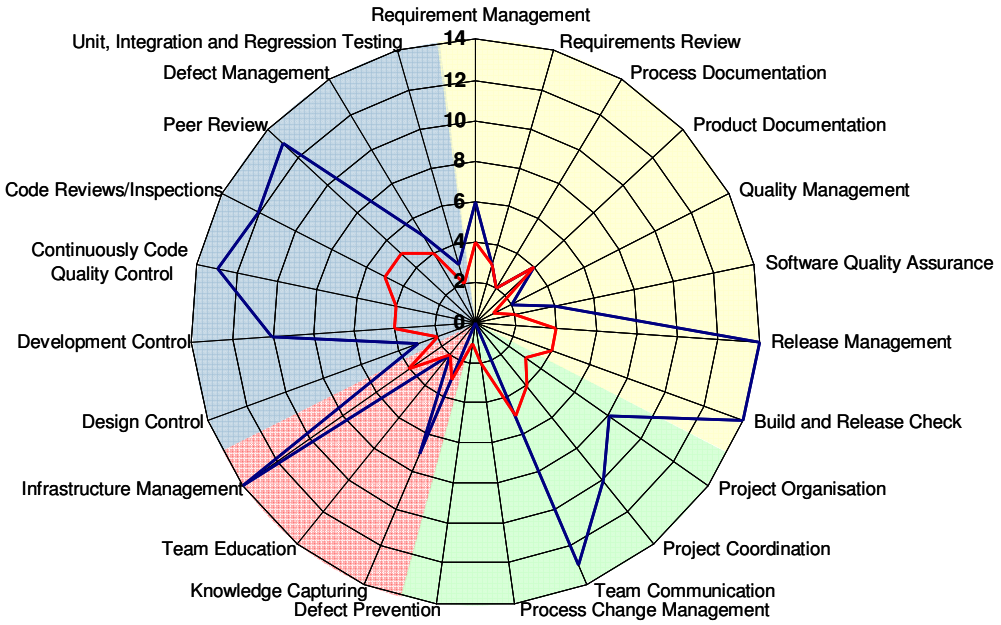

Figure 94. Analysis of Case Study Seven

8.3.8 Discussion of the Findings

In the following discussion, the term “literal” classifies the type of replication of the case studies (one to six), where similar results are predicted and a confirmation of theory is expected, while the term “theoretical” describes the type of replication, which shows contradictory results (case study seven).

The “literal” case studies distinguish different types of application, with mid- to high complexity and different types of motivation, such as community or commercially driven. The studies show, that high commercial interests influence processes related to management, organisation and personnel. For instance, paid full-time resources often take over a professional management role and offer a huge capacity increase.

Some commercially driven projects follow a hybrid approach, combining the traditional and OSS development model. An observed strength of these projects is their strict management focus on schedule, planning, steering and coordination and the active triggering of the OSS community. All “literal” projects benefit from an experienced team and direct communication within their community. It enables adjustments to the development direction and leads to the improvement of their product quality, as a result of efficient testing and debugging feedback. This reflects the high project success measurement results, concerning task completion, process stability and developer attraction.

Unlike the other cases, the “theoretical” case shows that processes are lacking, requirement management is mostly neglected, no management approach is visible and communication of required features and their status is insufficient (Otte *et al.,* 2008b). Reviews are informal and documentation is basic, messy, outdated, scattered and lacking in structure. Furthermore, the work environment is underdeveloped. The project success measurement reflects these shortcomings, resulting in a poor rating of developer attraction, task completion, process stability and programmer productivity.

The “literal” case study projects benefit from a mostly central requirement management, using an issue tracking tool or the project documentation. The integration of requirements follows defined processes, such as formal reviews by the core development team or the community, before new feature requests are chosen to be integrated into the design. The case studies have shown that documentation of processes commonly focus on style and coding guidelines. Complete product documentation concerning user- and technical documentation is mostly available. The projects often face problems over poor quality of user documentation, such as a lack of date-stamping. Moreover, tools to check documentation quality are absent.

Typically, knowledge capturing concentrates on mandatory documentation alongside the contributed source code. While many projects require the contribution of only highly experienced developers, some projects monitor developer skill levels and coach them accordingly.

All “literal” projects leverage tools, which are highly tailored to their needs and which support their development processes, while the “theoretical” case has an underdeveloped environment. With increasing project complexity, guidelines and process descriptions are mostly available. The analysis of the engineering processes shows that a comprehensive system design is part of the development. Results are documented, but sometimes projects fail to keep the documentation up-to date or are lacking design control.

The main strengths are continuous code quality checks, such as reviews, inspections, walkthroughs and peer reviews that are conducted by their communities. Beside manual checks, automated code style checkers are used to control the development activities. The applicability of pair programming faces problems in a distributed environment with different time zones. However, quality controls before commit or mandatory unit tests contribute to high code quality. Some projects perform integration tests prior to a new release, while some commercially driven projects in addition execute a complete regression test or conduct automated tests against design documents.

A key success factor is the enabling of the community to perform efficient testing or debugging, due to well-structured defect handling with classification and prioritisation. Furthermore, the achievement of a high reporting quality and defined testing processes, using a full testing suite, are essential components. These elements are missing in the “theoretical” case.

Some large projects assign responsibilities to a dedicated QA team for risk based testing, the approval of new releases and the coordination of defect fixing prior to a release. New releases are strictly managed, mostly following a feature-based approach, while often a time-based approach is used when it comes closer to a release. Typically, a feature freeze is conducted and release candidates are used for community testing. However, the “theoretical” case has insufficient release planning and no strict management.

In general, the case studies show that the management of quality targets is mostly neglected. Only a few projects consider code coverage analysis or focus on performance profiles or benchmarking. Quality targets, SQA activities as well as continuous process improvement approaches are informally managed, but there is strong motivation for defect prevention.

8.4 Project Success Measurement Results

The project success measurement model (introduced in section 7.6.4) was applied to each case study. In addition, an independent success estimation was conducted to compare the measured results. The independent project success assessment is based on the advancement of the processes, the observed product quality and the market satisfaction with the product.

8.4.1 Success Measurement Model Evaluation

In the following analysis, the validity of the project success measurement model against the general findings and research observations is discussed. The model is assessed and a conclusion suggested how the results can be applied for further analysis is drawn in the following sections.

The results of the success measurement are depicted in figure 95. Each graph presents the findings of every case study and shows the normalised results (R_n) of each question, while the graph (R_a) displays the average value.

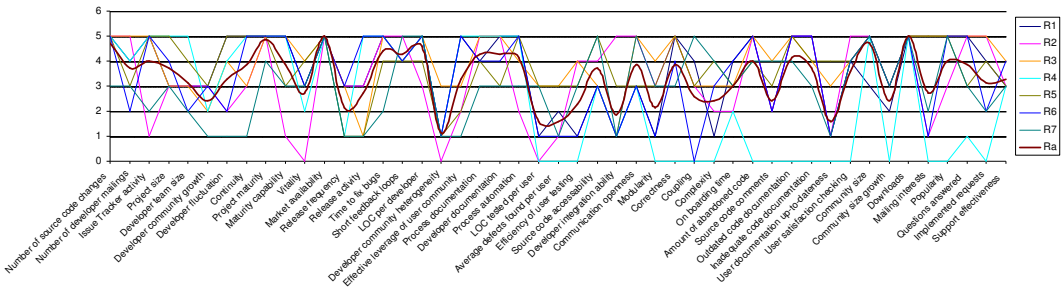

Figure 95. Success Measurement Results

Figure 96 shows the subsequent weightings regarding the importance of every question (W_n) scaled from 1-5, while the graph (W_a) depicts the average rating.

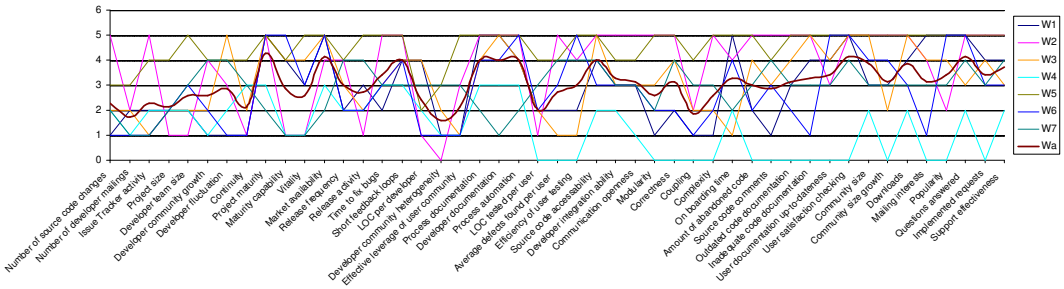

Figure 96. Success Measurement Weightings

The determination of the weighted project success score (PSS_W) is based on the total results and is used in the statistical analysis in section 8.4.2. The analysis of the results is clustered according to the areas “system creation and maintenance”, “system quality” and “sys-

tem use". Certain attributes are evaluated to verify the results and to determine the validity of the predefined possible answers. Special attention is given to the results, where ranking and average assessments deviate most.

8.4.1.1 System Creation and Maintenance Criteria

The sub-characteristics "activity/effort" are ranked above average; however, these criteria are rated less important. Respondents see project activity ratings as an effort indicator but one less significant for project quality evaluation. The analysis of the attribute "project size" shows similar results. In general, the "developer community growth" is rated poorly and the "continuity" is considered less essential. The respondents indicate that the growth in developer team size needs to comply with knowledgeable resources to achieve quality benefits for the project. According to the interviewees, the "activity/effort and attraction" criteria only reflect the quantity aspect, which does not express success factors of a project.

The pre-defined answer possibilities for the attribute "vitality" are either not understood correctly or respondents felt it meaningless.

Within the cluster "task completion", it is noticeable that the release frequency and release activity are assessed below average. The observed sample differs in their release intervals to the suggested values. This may question the ranking of the presetting.

Within the group "programmer productivity", the attribute "developer, community heterogeneity" is ranked very poorly and is not seen as a meaningful measurement criterion by the respondents. Furthermore, respondents weighted the attribute "LOC per developer" less important. The interviewees feel that the value varies tremendously and it is impossible to measure.

In the group "testing effectiveness", it is surprising that both attributes "LOC tested per user" and "average defects found per user" are regarded as inadequate. Again, these attributes are seen practically immeasurable. The respondents feel it is hard to determine how many use the software and how many lines they have tested. These criteria may only be estimated.

As expected, most of the projects show high values in terms of "project openness". However, the attribute "developer integration ability" is ranked low, which reflects that strict rules are applied mainly for developers new to a team.

8.4.1.2 System Quality Criteria

Within the cluster "code quality", the attributes modularity, coupling and complexity are assessed weakly. Respondents see that "modularity" is difficult to measure as it may differ within the parts of software itself. Furthermore, an important balance needs to be found between reusability and understandability aspects. Too many splits make it difficult to read the code, especially for newer contributors. On the other hand, reusability aspects ought to be considered, when necessary. The formula for the attribute "coupling" is felt as far too complex by the interviewees and difficult to obtain. Thus, these results may represent only estimations.

The attribute "complexity" is ranked below average and of less importance. Respondents think the assumption about the formula is incorrect. The interviewees argue if something is in line, well documented and only used once, then it is better to keep the source code together. It presents a high barrier to entry as well as a high ongoing challenge to developers if things splinter for no reason.

Within the cluster "documentation quality", it is interesting to note that the attribute "source code comments" is assessed averagely. Respondents believe it is difficult to measure this attribute, as too many comments can make codes more difficult to read. Moreover, a fine line needs to be drawn between the quality of the documentation as opposed to whether documentation "rambles" over many lines.

The attribute "user documentation up-to dateness" is considered an important criterion but the interviewees confirmed that difficulties exist to keep the documentation current, which explains the poor rating.

8.4.1.3 System Use Criteria

In general, the attributes in the area "system use" obtain a high weighting and ranking, except "community size growth". This may question the presetting of the pre-defined scale of this attribute or may simply show that the observed mature projects have a lower growth rate.

8.4.1.4 Validity of the Results

The success measurement model shows a normalised ranking of the sample, corresponding to the observations. Some pre-defined answers need to be questioned regarding validity and usefulness. However, due to the weighting of each question, the non-important ones have less consequence for the result. An additional analysis of the success measurement demonstrates that if the less or non-important questions are excluded from the sample, the weighted

project success value remains almost constant. Therefore, within this experiment all less important questions with an average weighting below the threshold of 3.0 are excluded from the sample. Figure 97 shows the normalised results (R_n) of each question based on selected project success measurement criteria which have an average weighting equal or greater than 3.0. The average value is displayed by the graph (R_a).

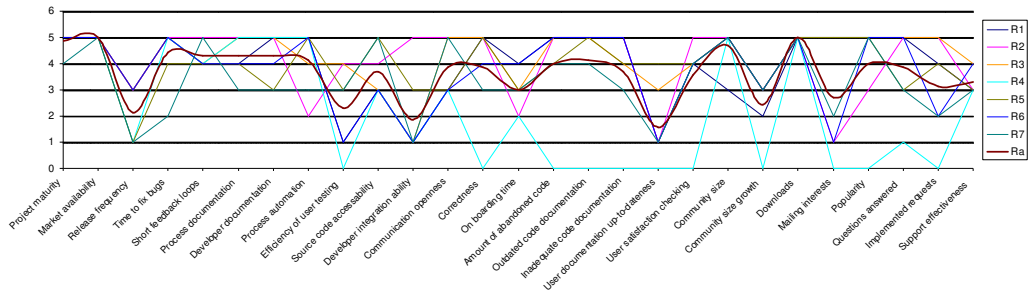

Figure 97. Selected Success Measurement Results

Their subsequent weightings are displayed in figure 98. It presents the importance of every question (W_n) scaled from 1-5, while the graph (W_a) depicts the average rating equal or greater than 3.0. These results are the basis for the determination of the selected weighted project success score (PSS_SEL) as analysed in section 8.4.2.

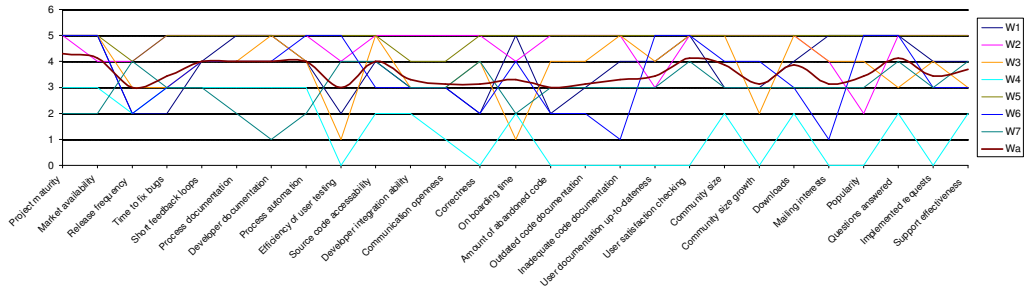

Figure 98. Selected Success Measurement Weightings

The analysis shows that the ratio of the average success value remains almost unchanged and a significant high positive correlation between the PSS_SEL and the PSS_W (r = .829, p = .011) can be found.

This means the determined projects success degree is independent of the excluded criteria. Both evaluations deliver comparable results, although some metrics ought to be excluded. Hence, the project success measurement model is regarded as a feasible assessment approach to determine the ratio of critical project success factors.

8.4.2 Statistical Analysis

The statistical analysis proves the correlation of process capability and the project success measurement results. The analysis was done using SPSS 16.0 to determine the linear regression and correlation (after Pearson). Table 17 summarises the results of the case studies and shows the process capability score (see Appendix B.4, table b.1 and table b.2) and the project success measurement scores (see Appendix B.5, table b.3) comprising total, average and weighted values.

Table 17. Case Study Analysis Summary

	CASE 1	CASE 2	CASE 3	CASE 4	CASE 5	CASE 6	CASE 7
Process Capability							
PCS Total	226	272	248	246	255	233	162
PCS Total weighted	893	1323	889	800	1187	935	634
PCS process average	*10.890*	*12.138*	*11.697*	*11.765*	*11.198*	*10.872*	*8.568*
PCS % coverage	61.413%	73.913%	67.391%	66.848%	69.293%	63.315%	44.022%
PCS % cov. weighted	**68.064%**	**75.860%**	**73.109%**	**73.529%**	**69.988%**	**67.951%**	**53.547%**
Success Measurement							
PSS pre weighted	3,698	3,396	4,047	3,936	3,698	3,544	2,982
PSS weighted	**3.884**	**3.701**	**4.167**	**4.082**	**3.779**	**3.556**	**2.926**
PSS average	3.638	3.319	3.957	3.931	3.745	3.426	2.979
PSS weighted selected*	**4.020**	**4.232**	**4.373**	**3.868**	**4.000**	**3.789**	**3.309**
PSS estimated	**8.000**	**7.500**	**8.500**	**8.000**	**7.000**	**7.500**	**5.000**

** The values are determined based on selected metrics according to their importance*

The process capability results are represented by the total score, the total score weighted, the average score per process, the percentage coverage of the model and the percentage coverage of the model according to its weighting factors. The weighted percentage PCS reflects the coverage of the model compared to actual results and considers the degree of importance for each process.

The project success measurement shows normalised values for each metric with its anticipated importance. Although this measurement approach offers an operable assessment approach, the model is seen as too static and does not reflect individual project requirements or needs. For instance, the release frequency may have a different degree of importance according to the maturity of a project. While an immature project forces frequent releases to push the development forward, a mature project may accept a lower frequency than suggested in the model. An individual weighting factor strengthens such characteristics, while less relevant criteria are of marginal consequence.

Within the further research, the measurement approach is revised. The measurement method (as introduced in section 7.6.4) remains unchanged, but instead of the static pre-defined weighting, the "anticipated importance" value is applied within the further analysis.

The statistical analysis focuses on the correlation of the weighted percentage PCS to the weighted PSS and the estimated PSS, as marked green in table 17. Moreover, an additional analysis based on selected success metrics (PSS_SEL) according to the selection criteria, as introduced in section 8.4.1.4, is applied. The pre-weighted value (PSS_PW) is shown for the sake of completeness.

In the subsequent analysis presented below, the following abbreviations are used:

- PCS_MCW – Process Capability Score percentage model coverage weighted
- PSS_PW – Project Success Score pre-weighted
- PSS_W – Project Success Score weighted (individual)
- PSS_EST – Project Success Score estimated
- PSS_SEL – Project Success Score weighted selected

The first analysis explores the correlation of the PCS_MCW to the determined PSS_W. The second analysis examines the correlation between the PCS_MCW and the independent PSS_EST. The aim of the second analysis is an alternative examination of the project success measurement scores in order to strengthen previous findings. The third analysis investigates the correlation of the PCS_MCW and the PSS_SEL, which reflects selected metrics of the project success measurement model based on their importance. The aim is to show, that independent of the selected criteria the model delivers similar results, which correlate with the process capability model.

A significant correlation in all analyses shows that the project success measurement depends on the process capability. Furthermore, it supports the validity of the project success measurement approach, as the measured PSS returns similar values to the independently estimated PSS.

8.4.2.1 First Analysis (PCS_MCW to PSS_W)

The first analysis explores the correlation of the process capability score percentage model coverage weighted (PCS_MCW) to project success score weighted (PSS_W). The model assumption is that the value of a dependent scale variable is based on its linear relationship to the predictors. Therefore, the linear regression method is used for hypothesis testing. Figure 99 displays a scatter plot of PCS_MCW to PSS_W and shows that a linear model is reasonable for these variables.

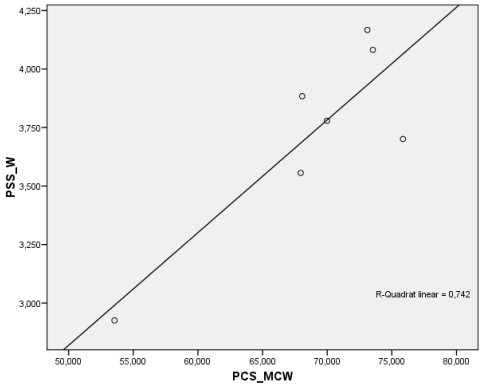

Figure 99. Scatter plot of PCS_MCW to PSS_W

Table 18 shows a significant high positive correlation between the PCS_MCW and the PSS_W (r = .862, p = .006). The R-Square displays the coefficient of determination, which is the squared value of the multiple correlation coefficients. It shows that 74.2% of the variation of project success is explained by the process capability.

Table 18. Model Summary – PCS_MCW to PSS_W

Model	R	R-Square	Adjusted R-Square	Std. Error of the Estimate
1	,862(a)	,742	,691	,229159

a Predictors : (Constant), PCS_MCW
b Dependent Variable: PSS_W

The Fisher-Test determines the significance of the regression. The significance value of the F statistic is less than 0.05 (as displayed in table 19). This rejects (p = .013) the null hypothesis, which indicates that no linear correlation exists.

Table 19. ANOVA – PCS_MCW to PSS_W

Model		Sum of Squares	Df	Mean Square	F	Significance
1	Regression	,756	1	,756	14,395	,013(a)
	Residual	,263	5	,053		
	Total	1,018	6			

a Predictors : (Constant), PCS_MCW
b Dependent Variable: PSS_W

Table 20 depicts the project success coefficient of the regression line. The coefficient shows that if the independent PCS_MCW increases about one unit, the dependent PSS_W increases about 0.48 units.

Table 20. Coefficients – PCS_MCW to PSS_W

Model		Un-standardised Coefficients		Standardised Coefficients		
		B	Std. Error	Beta	T	Significance
1	(Constant)	,412	,878		,469	,659
	PCS_MCW	,048	,013	,862	3,794	,013

a Dependent Variable: PSS_W

Figure 100 shows that neither the histogram nor the P-P diagram indicates that the normality assumption is violated. The analysis has shown that the dependent variable PSS_W can be explained by the PCS_MCW and a high correlation exists.

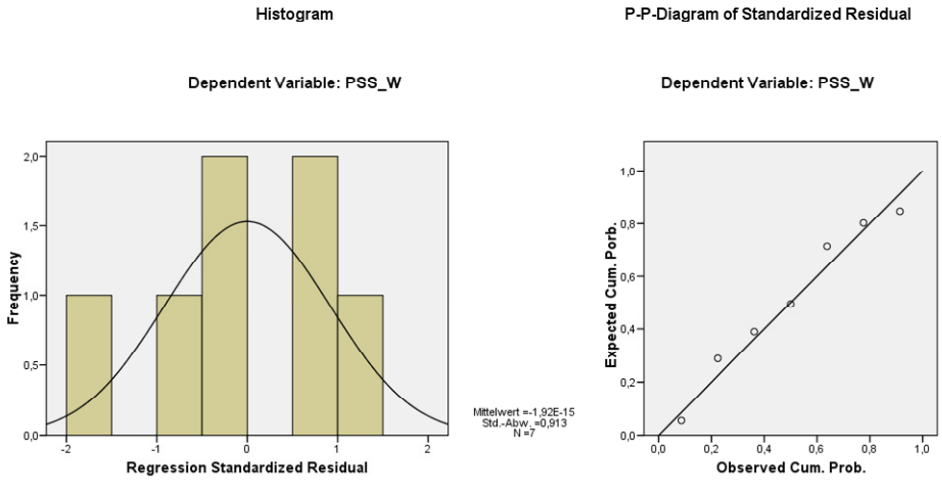

Figure 100. Histogram and P-P Diagram of PSC_MCW to PSS_W

8.4.2.2 Second Analysis (PCS_MCW to PSS_EST)

The second statistical analysis focuses on the correlation of the process capability score percentage model coverage weighted (PCS_MCW) to the estimated project success score (PSS_EST), showing a similar result.

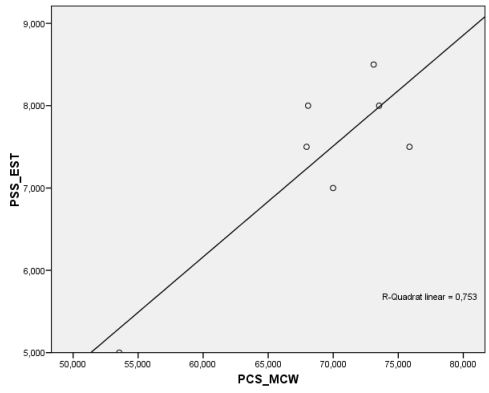

Figure 101. Scatter plot of PCS_MCW to PSS_EST

A high correlation is observed between PCS_MCW and PSS_EST (r = .867, p = .006) (see table 21).

Table 21. Model Summary – PCS_MCW to PSS_EST

Model	R	R-Square	Adjusted R-Square	Std. Error of the Estimate
1	,867(a)	,753	,703	,623641

a Predictors : (Constant), PCS_MCW
b Dependent Variable: PSS_EST

The R-Square displays that 75.3% of the variation of the estimated project success is explained by the process capability. The Fisher-Test results demonstrate a significant value (p = .011) (see table 22), which rejects the null hypothesis.

Table 22. ANOVA – PCS_MCW to PSS_EST

Model		Sum of Squares	df	Mean Square	F	Significance
1	Regression	5,913	1	5,913	15,202	,011(a)
	Residual	1,945	5	,389		
	Total	7,857	6			

a Predictors : (Constant), PCS_MCW
b Dependent Variable: PSS_EST

Table 23 shows the coefficient of the regression line, which means that if PCS_MCW increases about one unit the dependent PSS_EST increases about 0.135 units.

Table 23. Coefficients – PCS_MCW to PSS_EST

Model		Un-standardised Coefficients		Standardised Coefficients		
		B	Std. Error	Beta	T	Significance
1	(Constant)	-1,916	2,390		-,802	,459
	PCS_MCW	,135	,035	,867	3,899	,011

a Dependent Variable: PSS_EST

The histogram and the P-P diagram in figure 102 depict that the assumption of normality of the error term is not violated. The second analysis has shown that the dependent variable PSS_EST can be explained by the PCS_MCW and a high correlation exists.

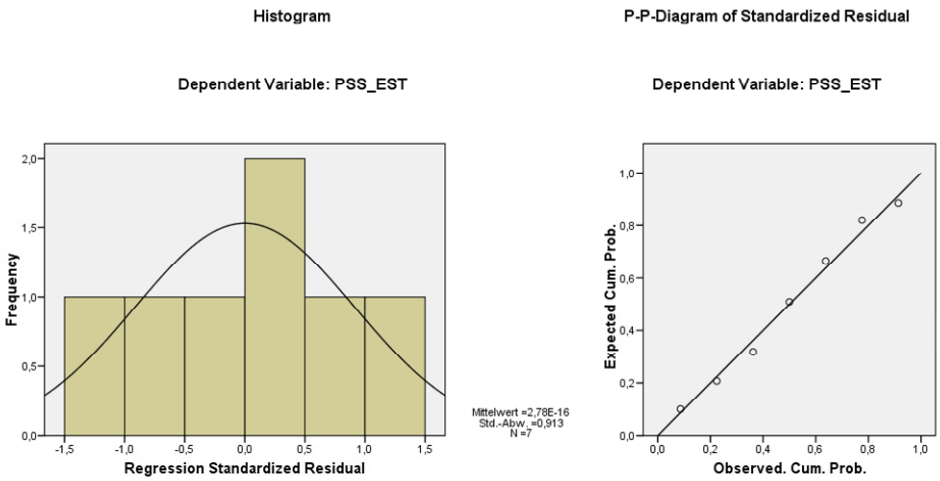

Figure 102. Histogram and P-P Diagram of PSC_MCW to PSS_EST

8.4.2.3 Third Analysis (PCS_MCW to PSS_SEL)

The third analysis compares the PCS_MCW and the selected weighted project success value (PSS_SEL) and delivers a similar result compared to the previous studies. A high correlation between PCS_MCW and PSS_SEL exists (r = .878, p = .009) (see Appendix B.6). According to the R-Square, 77.1% of the PSS_SEL is explained by the process capability. The Fisher-Test shows a significant result (p = .009). The coefficient of the regression line is 0.041 units and shows the increasing of PSS_SEL if PCS_MCW increases about one unit. The assumption of normality of the error term is not violated, as depicted in figure 103. The dependent variable PSS_SEL can be explained by the PCS_MCW, showing a high correlation.

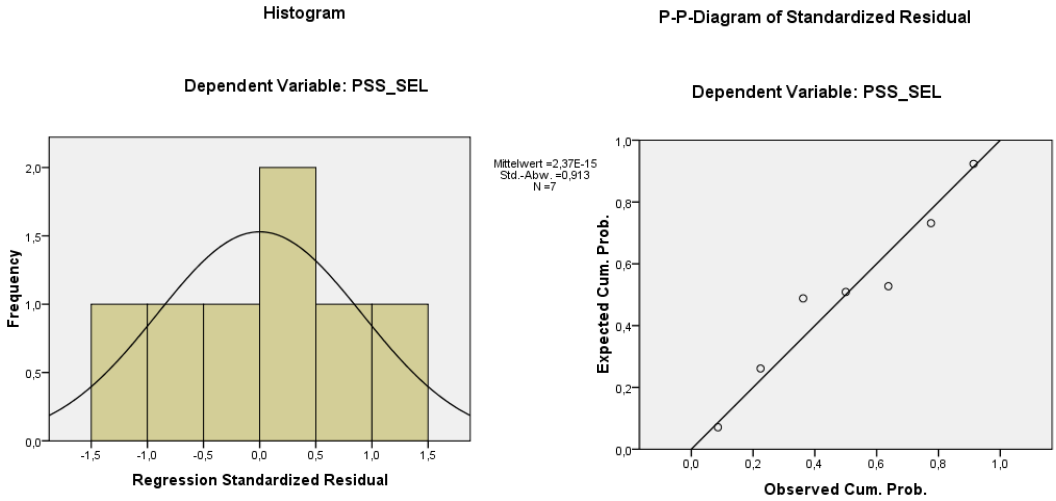

Figure 103. Histogram and P-P Diagram of PSC_MCW to PSS_SEL

8.4.2.4 Analysis Results

The statistical analysis confirms that a significant correlation between project capability and project success exists. The different project success scores, such as the estimated (PSS_EST), the selected (PSS_SEL) and the measured (PSS_W), can be explained by the project capability score (PCS_MCW). Moreover, a significant correlation between the estimated (PSS_EST) and measured project success score (PSS_W) (r = .945, p = .001) exists. This demonstrates the project success measurement model matches the estimated values, showing corresponding results.

8.5 Summary

The case study approach confirms the applicability of the QAfOSS model in applied OSS projects. The "literal" case studies show a high compliance with the QAfOSS process model, with minor deviations regarding quality management, process improvements and knowledge transfer processes. These findings correlate with the observed process capability and show a high compliance of the framework with existing processes, as depicted in figure 104. The "theoretical" case study deviates from the QAfOSS model, especially in relation to management, organisational and resource related activities are beyond the process targets.

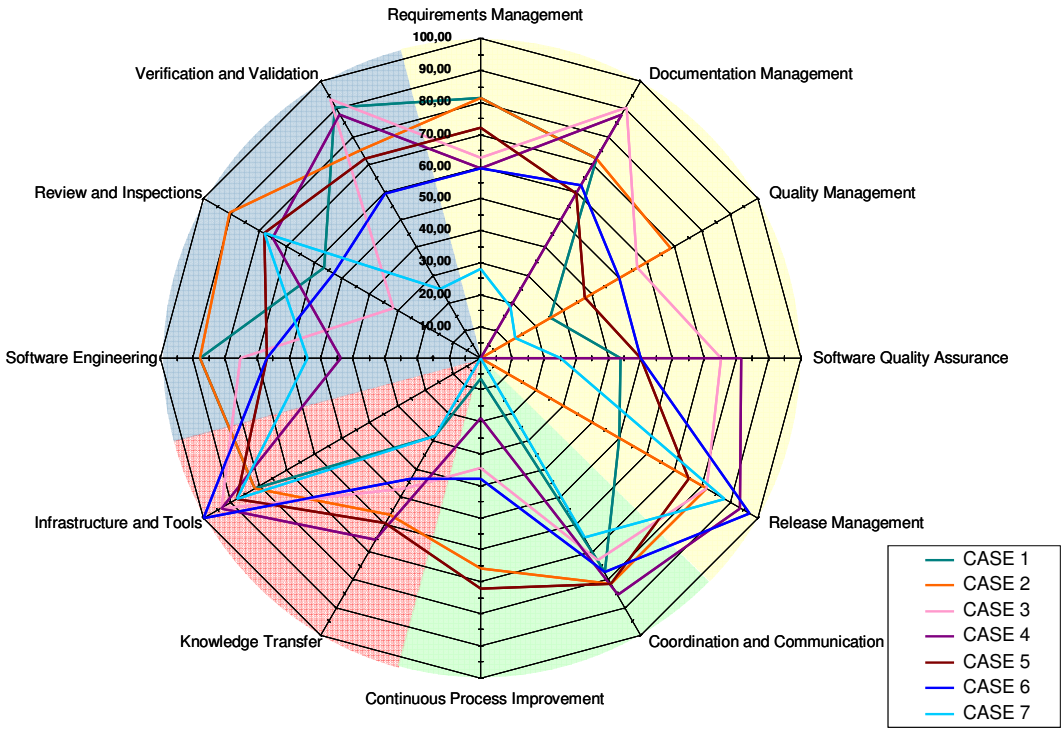

Figure 104. Case Study Process Capability Determination Summary

Although the measurement model has some shortcomings, as discussed in section 8.4.1 and 9.2, the project success measurement approach delivers a feasible ratio of success factors. In

some cases, the success measurement criteria do not apply and difficulties exist to define commonly accepted scales. Due to the weighting of each metric, it upgrades important criteria, leading to levelled results. The experiment has shown that if metric formulae are applied similarly by all projects, the ratio will remain almost unchanged. In addition, the analysis of the alternative project success estimation confirms the findings. Although the measurement approach has certain limitations, it constitutes an appropriate basis for comparisons when applied in the manner defined.

High process coverage can be found in the group of "successful" projects, while the less "successful" project shows deficits in management, organisation and engineering, resulting in a lower project success score. The statistical analysis confirms a significant correlation of the process capability and the success measurement score. The independent project capability scope can be explained by the dependent project success value. This means that projects, which obtain a high project capability score, can achieve a high success value. If these projects gain their success level mainly due to the implementation of the suggested process model, it could be concluded that the proper implementation of the model always leads to success. This argument is misleading, because SQA processes are the focus of the QAfOSS framework and further criteria that are of importance for a project success are neglected, such as social constraints. In conclusion, project success depends on the integration of management, organisation or social considerations, beside the establishment of QA processes. Nevertheless, the high correlation and the qualitative findings show that the observed projects confirm the applicability of the framework supporting SQA under the OSSD.

9 Summary and Conclusions

Within the final chapter, the corollaries and consequences of the research are discussed. The critical discussion focuses on the findings and reviews of the research approach. The key outcomes and the contribution to knowledge are highlighted. Finally, a summary of the research limitations is provided and further research perspectives are shown.

9.1 Research Summary

The OSSD model offers many advantages. One of the most important aspects is the collaborative distributed development approach that leverages the communities efficiently for testing, debugging or QA. The OSSD model applies several software development practices, while the method itself is a unique approach that follows clearly defined rules. The OSSD model is not only used in purely free projects, because nowadays many companies integrate it into their existing development methods. Thus, such projects take advantage of the OSS while continuing their traditional model.

Projects face the question of how to deliver constant high software quality. In order to use OSS products it is necessary that users and companies can rely on them and their development practices. Although many OSS products claim to be successful in the market, the literature review identifies several quality problems concerning the development processes. It has been argued that empirical evidence of applied key processes in mature projects is lacking. Moreover, a common quality assurance model that supports the OSSD is absent and the need for further investigation has been explained.

The major objective of this thesis is the investigation of SQA processes under the OSSD. Thus, two central research questions were posed. The first research question examines: *"how is Software Quality Assurance in OSS projects applied and what key practices characterise mature projects?*"

The OSSD is characterised as an iterative, parallelised, developer-driven development approach with voluntary collaborating developers working in a geographically distributed environment through web-based technologies. Recent literature findings contribute to the understanding of SQA processes under the OSSD, which have been the subject of research by e.g. Zhao and Elbaum (2000, 2003), Halloran and Scherlis (2002), Koru and Tian (2004), Michlmayr (2005) or Aberdour (2007). Although the OSSD model differs from traditional approaches, its QA methods have many similarities combined with unique strengths. Testing and debugging processes benefit from large communities in relation to test efficiency. Under the OSSD, the testing scope is expanded and may cover multiple system environments. The

bug reporting quality contributes to effective defect handling. Collaboration and development processes benefit from a highly sophisticated tool support. Several shortcomings of SQA processes under the OSSD have been discussed. The documentation is often neglected, the design is lacking, testing is done informally, code remains unmaintained or measurable quality goals are often missing.

In order to extend existing knowledge, a further investigation of applied QA methods was conducted in a uniform analysis using the survey method. The intention was to gather actual data from a changing OSSD model and to verify the findings. While the previous studies of various authors analyse different target groups, the survey explored a broad range of practices within one dedicated target group. Thus, it enabled a comparison with the literature findings. The first analysis focused on the exploration of QA practices and an in-depth evaluation of mature projects identified key processes. This research contributes to knowledge with a holistic view of applied QA practices and development processes. While previous work often criticised OSS as unstructured or unsystematic (Massey 2003), the survey findings show that mid- to large projects follow common processes and methods that contribute to SQA. This provides evidence for Raymond's (2001) lifecycle. The examination of key QA practices shows that quality largely depends on sustainable communities, well structured testing and defect handling processes, high applicability of reviews and inspections, suitable documentation, process documentation, good organisation and clear coordination, effective communication and efficient tool support to manage complex processes. The survey provides evidence that mature projects consider code modularity during design, have a stricter quality control before code commit and spend more time on testing, which is highly efficient. The defect handling is well structured covering many topics and the documentation approach supports the knowledge transfer. Internal communication seems well developed, organisations are well defined and highly supported by tools. SQA processes are more often implemented in larger projects.

The analysis of "successful" mature projects explores applied key processes that support SQA under the OSSD and contributes to knowledge supported by empirical evidence. The survey comprises a broad sample and combines quality criteria and project success measures. The detailed analysis focuses on general, organisational and human resource issues, as well as processes and methods, testing, defect handling and quality assurance tasks. The research identifies important criteria that contribute to SQA, such as requirement management, documentation, coordination aspects, project attraction, need for initial planning and design, quality control processes, testing, release management, quality management and SQA. The findings reveal a correlation of project success and applied development processes exists, but not

necessarily any causalities. Moreover, it confirms recent knowledge, enriches the understanding of applied QA practices and constitutes a basis for further research.

The second research question explores *"can a quality assurance framework contribute to the improvement of the quality assurance activities in the OSSD?*" First, the research investigated the development of a QA framework. The extensive literature review of product- and process-oriented quality approaches contributed to the definition of the research fundamentals. The research follows the assumption that the quality of the product depends on the processes used to build it. However, mature processes do not imply well assured product quality. Therefore, a process model approach is chosen, which considers product quality characteristics. The ISO 12207 life cycle model and the CMM constitute a basis for the development of the process model, while the ISO 9126 standard is applied for product quality measurement.

As an underlying structure, a generic framework was developed and the identified key processes were consolidated. The main contribution to knowledge of this research is a novel framework with a focus on OSS product and process quality. The process model is based on the following assumptions:

- Requirement management needs to reflect evolving feature requests and sets the foundation for development, testing and acceptance of the product.
- Documentation management affects the knowledge transfer and is a key factor in sharing, enriching and capturing know-how.
- Strict project coordination is required to enforce the management of activities and to ensure collaboration.
- A suitable project organisation and authorisation concept following the OSSD model is mandatory. For instance, the definition of access rights to the repository in order to ensure reviews before code commit.
- Communication processes and a suitable infrastructure with integrated tools are key factors to enable the distributed development approach.
- A project requires a certain degree of 'attraction' to attain the critical mass of voluntary contributors to utilise the advantages of large teams.
- The development processes ought to consider initial planning and design activities to avoid quality issues when the requirements evolve.
- Quality control processes to check the adherence to standards and the use of reviews are of major importance to assure quality.
- Projects need to set up reasonable testing processes and defect management to exploit their community size effectively and to benefit from user suggestions.
- Release management becomes an integral element for QA. It comprises scope management, triggers testing activities and forces the establishment of release candidates to assure a wide verification.
- Quality management is required to define measures and to control specific quality targets.

- SQA processes obtain a central role to achieve the implementation and the effective interaction of the QA framework elements.

A process measurement approach was introduced to assess the implementation level and it describes the process capability of an OSS project. The QAfOSS framework summarises "best practices".

This research contributes to knowledge by an applied success measurement model suggesting metrics tailored for OSS projects. The project success measurement approach offers a success evaluation based on the work of Crowston *et al.* (2006) and completes the method for OSS projects. The approach was introduced to assess the process framework. While the correlation of both methods is used to prove the QAfOSS framework, each method can stand alone. OSS projects can apply the process model to evaluate their current approach and to identify weaknesses and shortcomings. The success measurement approach offers independent benchmarking for a project. The limitations and the need for further research are presented in section 9.2.

The applicability of the framework was proven by the correlation of measured process capability and project success. A positive correlation showed that the suggested QA practices of the QAfOSS framework efficiently contribute to SQA within the projects. The application of the model in cross-case studies confirms a significant correlation and clearly indicates a high model coverage with "successful" projects. The "theoretical" case provides a critical examination of a less successful project against the requirements of the QA model. The study has shown that the QAfOSS framework can support adequate software quality assurance under the OSSD.

9.2 Research Limitations and Future Directions

Within the applied chain of nomothetic and ideographic research methods, different limitations of the results are discussed. First, the reliability of the survey findings and the completeness of identified key processes are explored. Second, the totality of the framework processes, the measurement model and its applicability to support SQA are examined. Third, the validity of the case study findings and the success measurement results are shown.

Survey Findings

Although, the survey research follows the approach of Malhotra and Grover (1998), the research faces some limitations. The usability of the findings depends on the validity of the survey instrument, reaching the target group and retrieving valid responses. Poorly understood questions or inappropriate scales might bias the results. In order to achieve a higher significance, the survey is based on an interpretative tradition. A qualitative analysis is per-

formed by using triangulation. The qualitative findings and conclusions are drawn from empirical and interpretative data. The intention of the survey was to incorporate success metrics in a set of QA criteria. Although the survey provides a large amount of descriptive data about QA practices, a classification of "successful" projects based on the success metrics is not accomplishable. The defined threshold regarding time in the market, release version and project team size predicates the degree of maturity, but does not subsequently imply project success. In consequence, a widespread analysis of success measurement criteria following the suggested model by Crowston *et al.* (2006) should be subject to future research.

Framework Processes

The development of the process framework is based on the consolidated findings from the literature and survey research. The model is a synthesis of common practices, such as ISO12207 and identified key processes. A high degree of coverage is anticipated, originating as it does from proven process standards. Although several interviews have been conducted to validate the model, its completeness cannot be claimed. Further research is required to review the integrity of the model and to adapt future practices of the constantly altering development approach.

Process Capability Measurement

The process capability measurement approach uses product quality characteristics, following Satpathy *et al.* (2000), to assess the degree of process fulfilment. Although the assessment method has many similarities with proven models, such as OSMM or BRR, this method faces certain limitations. While the QAfOSS process model is static and defines the expected outcome and necessary practices, in OSS projects the boundary between processes are often blurred or practices are informally applied. Therefore, a transition to the QAfOSS processes is required. Sometimes processes are applied across sectors and the assessor has to evaluate the fulfilment of each practice. Thus, an assessor must apply identical process benchmarks to achieve comparable results. In consequence, OSS process assessments with non harmonised benchmarks may face certain limitations regarding their general comparability. Further studies may refine the assessment criteria of the process model in order to develop commonly accepted benchmarks. A subdivision of practices with respective assessment criteria may improve the comparability of independent assessments.

Success Measurement

The proposed success measurement model continues the research with "success" evaluation, as primarily applied during the survey research. While the survey results do not allow a distinct classification of project success, the revised model of Crowston *et al.* (2006) is supple-

mented with metrics that deliver normalised results for an improved appraisal. The formulae originate from literature (referring to Appendix B.3), but new scales have been introduced to normalise the results.

The measurement approach faces certain limitations due to the pre-defined answer types with standardised results. The responses to the full survey suggested that the result could be biased due to errors in the formula, incorrect success assumptions of the predefined values or too complex questions, which may not be understood correctly by the interviewees. Measurement criteria, that are found meaningless by the respondents, ought to be excluded from the measurement approach, such as the attribute "vitality". In some cases, the pre-ranking of results does not fit easily. Normalised pre-defined answers are useless if qualitatively similar results vary between the projects. For instance, the release frequency can deviate tremendously between projects, without qualitative effect. Thus, it makes success measurement all but impossible. Some questions are not answered in the case studies. Since these questions have been excluded from analysis, evaluation is limited, which may compromise the individual result. The success measurement approach has shown that difficulties exist in defining common success metrics. Further research is required to validate the suggested metrics and to establish a widely accepted measurement approach.

Case Study Results

The qualitative analysis of the case study results show limitations regarding method, approach and applied analysis tools. The information retrieval based on a questionnaire faces similar problems as discussed in the survey method (chapter 4.3.1). The results can vary, as the objectivity depends on the view of each participant and it cannot be guaranteed that the same standards are applied. The evaluation level differs arising from the participant's knowledge and bias could lead to distortions in the result. Beside an empirical measurement, the qualitative assessment delivers clearer insights and helps validate the assessment instrument. The case studies were conducted with a limited number of organisations. Although the findings deliver useful results, the sample is small and outliers may overly influence the statistical analysis.

The exploration of hybrid projects is considered of great importance. These approaches combine traditional methods with the OSSD model. The analysis of applied processes may be the subject of further research. A broad study could provide further evidence about best practices, improving SQA and may lead to a validation and enhancement of the QAfOSS model.

9.3 Overall Conclusion

This research investigates QA practices under the OSSD and contributes to knowledge by the development of the QAfOSS framework. A number of conclusions are drawn from the research.

First, the development lifecycle is supported by the process model and an accompanying product quality assessment is mandatory to assure quality. Following the assumption that product quality depends on the key processes used to develop it, several important findings from OSS projects can be derived from the development of the process model:

- SQA is achieved by collaborative user testing, intensive debugging and direct user suggestions. Such parallelised processes mainly contribute to SQA and leverage from integrated tools.
- SQA depends on process quality and the involvement and attraction of the contributors. Therefore, a balanced interaction of rewarding, high transparency of processes, defined organisation and a simple information access is necessary.
- The enabling of the community to contribute efficiently to development processes is of major importance. Thus, foundation elements such as an appropriate documentation, communication and project organisation are essential.

Second, the case studies reveal that the process model and the assessment method are applicable to OSS projects. Although some difficulties exist, the success measurement approach delivers an appropriate classification and may provide other researchers with a basis for further investigations. However, further qualitative assessment is suggested for validation. The process capability determination approach shows a usable degree of process implementation and has been verified within the case studies. The QAfOSS model follows the assumption, that the higher the QAfOSS process capability, the higher the probability of a project success. Nevertheless, purely implemented processes that are considered as an improvement, do not subsequently lead to success. The achievement of a high software quality depends on several aspects, such as sociological factors, environmental issues, human factors, project attraction, management ability and QA relevant processes. The QAfOSS framework contributes to an improvement of QA processes, while the collaboration of all aspects is required to achieve high product quality. OSS projects need high self-initiative, good management discipline, quality awareness, developer motivation, integrative knowledge of tools and development practices. Further research needs to incorporate these aspects and explore the implementation of the QAfOSS model in longitudinal studies.

10 References

ABBOTT, L. (1955) *Quality and Competition.* New York: Columbia University Press, pp. 126–127.

ABERDOUR, M. (2007) Achieving Quality in Open Source Software. *IEEE Software*, **24(1)**, pp. 58–64.

ABRAHAMSSON, P., SALO, O., RAONKAINEN, J., and WARSTA, J. (2002) *Agile software development methods: Review and Analysis* [online]. Espoo, Finland: Technical Research Centre of Finland, VTT Publications 478, [cited 11th July 2007]. <http://www.vtt.fi/inf/pdf/publications/2002/P478.pdf>.

AHMAD, U. and LODHI, F. (2006) An Open Source Software Development Model. *in Proceedings of ICOST-2006, International Conference on Open-Source Technologies, December 28-29, 2006, Lahore, Pakistan.*

ANDERSEN, N., KESING, F., LUNDIN, J., MATHIASSEN, L., MUNK-MADSEN, A., RASBECH, M. and SORGAARD, P, (1990) *Professional Systems Development: Experience, Ideas and Action*. Hemel Hempstead: Prentice-Hall.

ANKOLEKAR, A., HERBSLEB, J. D. and SYCARA, K. (2003) Addressing Challenges to Open Source Collaboration With the Semantic Web. *in Proceedings of Taking Stock of the Bazaar: The 3rd Workshop on Open Source Software Engineering*, the 25th International Conference on Software Engineering (ICSE), May 3-10, 2003, Portland OR, USA, pp. 9–13.

ARI, I. (2001) *Quantitative Analysis of Open Source Software Projects* [online.] [cited 10th December 2007]. <http://www.soe.ucsc.edu/~ari/ari-quan-OSS.pdf>

ATA, C., GASCA, V., GEORGAS, J., LAM, K. and ROUSSEAU, M. (2002) *Open Source Software Development Processes in the Apache Software Foundation* [online]. [cited 18th April 2007]. <http://www.ics.uci.edu/~michele/SP/index.html>

BAKER, E.R. and FISHER, M.J. (1982) A Software Quality Framework. *Journal of Defense Systems Acquisition Management*, **5(4)**, Fort Belvoir, VA: Defense System Management College.

BAKER, E.R. and FISHER, M.J. (1999) Software Quality Program Organisation. *in* SCHULMEYER, G.G. and MCMANUS, J.I. (eds.) *Handbook of Software Quality Assurance.* 3rd ed., Upper Saddle River, NJ: Prentice-Hall, pp. 115–145.

BASILI, V.R. (1985) *Quantitative evaluation of software methodology*. College Park, Md: University of Maryland.

BASILI, V.R., CALDIERA, G. and ROMBACH, H.D. (1994a) Experience Factory. *in* MARCINIAK, J.J., (eds.), *Encyclopedia of Software Engineering, Volume 1*. John Wiley & Sons, pp. 469–476.

BASILI, V.R., CALDIERA, G. and ROMBACH, H.D. (1994b) The Goal Question Metric Approach. *in* MARCINIAK, J.J., (eds.), *Encyclopedia of Software Engineering, Volume 1*. John Wiley & Sons, pp. 528–532.

BASILI, V., BRIAND, L., CONDON, S., YONG-MI, K., MELO, W. and VALEN, J., (1996) Understanding and predicting the process of software maintenance releases. *in Proceedings of the 18th International Conference on Software Engineering,* pp. 464–474.

BASKERVILLE, R. (1999) Investigating Information Systems with Action Research [online]. *Communications of the AIS*, **2(19)**, [cited 28th June 2007]. <http://www.cis.gsu.edu/~rbaskerv/CAIS_2_19/CAIS_2_19.html>

BEHLENDORF, B. (1999) Open source as a business strategy. *in* DIBONA, C. (eds.) *Open Sources: Voices from the Open Source Revolution*, 1st ed., O'Reily, pp. 149–170.

BENBASAT, I., GOLDSTEIN, D.K. and MEAD, M. (1987) The Case Research Strategy in Studies of Information Systems. *MIS Quarterly*, **11(3)**, pp. 369–386.

BERANDER, P, DAMM, L.O., ERIKSSON, J., GORSCHEK, T., HENNINGSSON, K., JÖNSSON, P., KAGSTRÖM, S., MILICIC, D., MARTENSSON, F., RÖNKKÖ, K., TOMASZEWSKI, P. (2005) *Software quality attributes and trade-offs.* Blekinge Institute of Technology, pp. 3–19.

BERRY, D.M. (2004) Internet research: privacy, ethics and alienation: an open source approach. *Internet Research,* **14(4)**, pp. 323–332.

BICEGO, A., KHURANA, M., and KUVAJA, P. (1998) BOOTSTRAP 3.0 – A SPICE conformant software process assessment methodology. *in* HAWKINS, C., ROSS, M. and STAPLES, G., (eds.), *Software Quality Management VI – Quality Improvement Issues.* London: Springer-Verlag, pp. 26–37.

BOEHM, B.W., BROWN, J.R., KASPER, H., LIPOW, M., MACLEOD, G. and MERRITT, R. (1978) *Characteristics of software quality*. TRW series of software technology, v. 1., Amsterdam: North-Holland Pub. Co.

BOEHM, B. (2002) Get Ready For The Agile Methods, With Care. *IEEE Computer*, **35(1)**, pp. 64–69.

BONOMA, T.V. (1985) Case Research in Marketing: Opportunities, Problems, and a Process. *Journal of Marketing Research*, **22(2)**, pp. 199–208.

BROOKS, F.P. (1995) *The mythical man-month: essays on software engineering*. Reading, Mass: Addison-Wesley Pub. Co.

BRR (2006) *Business readiness rating - a framework for evaluating open source software* [online]. BRR, [cited 10th October 2007]. <http://www.openbrr.org/wiki/index.php/Home>

BURRELL, G. and MORGAN, G. (1979) *Sociological Paradigms and Organizational Analysis.* London: Heinemann.

CAPILUPPI, A., LAGO, P., MORISIO, M. (2003) Evidences in the evolution of OS projects through changelog analyses. *in 3rd Workshop on Open Source Software Engineering*, the 25th International Conference on Software Engineering (ICSE), May 3-10, 2003, Portland OR, USA

CHECKLAND, P.B. (1981) *Systems Thinking, System Practise*. Chichester: Wiley.

CHO, C.K. (1980) *An Introduction to Software Quality Control.* New York: John Wiley & Sons, Inc.

CHO, C.K. (1987) *Quality Programming: Developing & Testing Software with Statistical Quality Control.* New York: John Wiley & Sons, Inc.

CHRISSIS, B.M., KONRAD, M. and SHRUM, S. (2003) *CMMI: Guidelines for Process Integration and Product Improvement.* Addison-Wesley Professional, pp. 11–74.

CMMI (2002) *Capability Maturity Model Integration (CMMISM), Version 1.1*. Carnegie Mellon - Software Engineering Institute, CMU/SEI-2002-TR-012.

CMMI (2006) *Capability Maturity Model Integration for Development, Version 1.2*. Carnegie Mellon - Software Engineering Institute, CMU/SEI-2006-TR-008.

COCKBURN, A. and HIGHSMITH, J. (2001) Agile Software Development: The People Factor. *IEEE Computer*, Nov. 2001, pp. 131–133.

COCKBURN, A. (2002) *Agile Software Development*. Boston: Addison-Wesley.

CONSTANTINE, L. (2001) Methodological Agility. *Software Development*, June 2001, pp. 67–69.

COTE, M.A., SURYN, W., MARITIN, R.A. and LAPORTE, C.Y. (2004) Evolving a Corporate Software Quality Assessment Exercise: A Migration Path to ISO/IEC 9126. *Software Quality Professional,* **6(3)**, pp. 5–7.

CRAIG SMITH, N. (1990) The case study: A useful research method for information management. *Journal of Information Technology*, **5**, pp. 123–133.

CROSBY, P. (1979) *Quality is Free: The Art of Making Quality Certain*. New York: New American Library.

CROWSTON, K. and HOWISON, J. (2005) The social structure of free and open source software development [online]. *First Monday*, **10(2),** [cited 19th April 2007]. <http://www.firstmonday.org/issues/issue10_2/crowston/index.html>

CROWSTON, K., HOWISON, J., ANNABI, H. (2006) Information systems success in free and open source software development: theory and measures. *Software Process: Improvement and Practice - Special Issue on Free/Open Source Software Processes,* **11(2)**, pp. 123–148.

DEBOU, C. (1999) Goal-based Software Process Improvement Planning. *in* MESSNARZ, R. and TULLY, C. (eds.) *Better Software Practices for Business Benefits: Principles and Experience*. IEEE Computer Society Press, pp. 107–150.

DELONE, W.H. and MCLEAN, E.R. (1992) Information Systems Success: The Quest for the Dependent Variable, *Information Systems Research*, **3(1)**, pp. 60–95.

DELONE, W.H. and MCLEAN, E.R. (2002) Information systems success revisited. *in Proceedings of the 35th Hawaii International Conference on System Sciences*. Volume 8, HICSS '02, IEEE Computer Society.

DELONE, W.H. and MCLEAN, E.R. (2003) The DeLone and McLean Model of Information Systems Success: A ten-Year Update. *Journal of Management Information Systems*, **19(4)**, pp. 9–30.

DEMING, W.E. (1988) *Out of the crisis: quality, productivity and competitive position.* Cambridge Univ. Press.

DE OLIVIERA, K.M., ROCHA, A.R., WEBER, K.C. (2002) Workshop on Software Quality. *in Proceedings of the 24th International Conference on Software Engineering*, New York: ACM Press, pp. 671–672.

DIBONA, C., OCKMANN, S. and STONE, M. (1999) *Open Sources - Voices from the Open Source Revolution.* O'Reilly.

DIETZE, S. (2005) Agile Requirements Definition for Software Improvement and Maintenance in Open Source Software Development. *in Proceedings of SREP'05, 2005, August 29-30, Paris, France.*

DILLMAN, D. (1978) *Mail and Telephone Surveys: The Total Design Method*. New York.

DILLMAN, D. (2000) *Mail and Internet Surveys. The Tailored Design Method*. New York.

DINKELACKER, J., GARG, P.K., MILLER, R. and NELSON, D. (2002) Progressive Open Source. *in Proceedings of the International Conference on Software Engineering (ICSE'02),* Orlando: ACM Press, pp. 177–184.

DROMEY, R.G. and MCGETTRICK, A.D. (1992) On Specifying Software Quality. *Software Quality Journal*, **2**, pp. 45–74.

DUIJNHOUWER, F.W. and WIDDOWS, C. (2003) *Open Source Maturity Model* [online]. Seriouslyopen.org (2007), [cited 30th April 2007]. <http://www.seriouslyopen.org/nuke/html/modules/Downloads/osmm/GB_Expert_Letter_Open_Source_Maturity_Model_1.5.3.pdf>.

DUNN, R. (1990) *Software Quality Concepts and Plans*. Prentice Hall, p.11.

EDWARDS, C.D. (1968) *The meaning of quality*, Quality Progress, October, pp. 35–40.

EDOS (2007) *Environment for the development and distribution of open source software (EDOS) project* [online]. EDOS, [cited 10th October 2007]. < http://www.edos-project.org/xwiki/bin/view/Main/>.

EL-EMAN, K. (2001) Ethics and Open Source. *Empirical Software Engineering,* **6(4),** pp. 291–292.

FAGAN, M. (1976) Design and code inspections to reduce errors in program development. *IBM Systems Journal*, **15(3),** pp. 182–211.

FAGAN, M. (1986) Advances in software inspections. *IEEE Transactions on Software Engineering*, **12(7)**, pp. 744–751.

FAIRLEY, R. (1997) *Software Quality Engineering Course Notes*. 16. April 1997.

FEIGENBAUM, A.V. (1991) *Total Quality Control.* (3rd ed.), McGraw-Hill, p. 1.

FELLER, J. and FITZGERALD, B. (2000) A framework analysis of the open source software development paradigm. *in Proceedings of the twenty first international conference on Information systems ICIS 2000, Brisbane, Queensland, Australia*, pp. 58–69.

FENTON, N. (1991) *Software Metrics: A Rigorous Approach.* Chapman and Hall.

FENTON, N. (1996) Do standards improve quality: A counterpoint. *IEEE Software*, **13(1)**, pp. 23–24.

FENTON, N., E., PFLEEGER, S.L. (1997) *Software Metrics: A Rigorous and Practical Approach.* 2nd ed., Boston: PWS Publishing, p. 5.

FRANKL, P., HAMLET, R., LITTLEWOOD, B. and STRIGINI, L. (1998) Evaluating testing methods by delivered reliability. *IEEE Transactions on Software Engineering,* **24(8),** pp. 586–601.

FSF (2005) *The free software definition* [online]. Free Software Foundation (2005), [cited 18th March 2005]. <http://www.gnu.org/philosophy>

FSF (2007) *Why "Open Source" misses the point of Free Software* [online]. Free Software Foundation (2007), [cited 11th April 2007]. <http://www.gnu.org/philosophy/open-source-misses-the-point.html>

FOWLER, F.J. (1984) *Survey Research Methods*. Thousand Oaks: CA, Sage.

GABLE, G.G. (1994) Integrating case study and survey research methods: an example in information systems. *European Journal of Information Systems*, **3(2)**, pp. 112–126.

GALLIERS, R. (1985) *In search of paradigm for information systems research. in* MUMFORD, E., HIRSCHHEIM, R., FITZGERALD, G. and WOOD-HARPER, T. (eds.) *Research Method in Information Systems.* North-Holland: Elsevier Science Publishers, pp. 271–284.

GALLIERS, R. (1992) *Choosing Information Systems Research Approaches. in* GALLIERS, R. (ed.) *Information Systems Research: Issues, Methods and Practical Guidelines*. Blackwell Scientific Publications.

GARVIN, D. (1984) *What Does "Product Quality" Really Mean?*. Sloan Management Review, Fall 1984, pp. 25–45.

GENTLEMAN, W.M. (1996) *"If Software Quality is a Perception, How Do We Measure IT?"* Institute for Information Technology, National Research Council of Canada, Ottawa, p. 3.

GILB, T. (1977) *Software metrics*. Winthrop computer systems series. Cambridge, Mass: Winthrop Publishers.

GILLIES, A.C. (1997) *Software Quality: Theory and Management. 2nd ed.,* International Thomson Computer Press.

GLASS, R. (2001) Is open source software more reliable? An elusive answer. *The Software Practitioner*, **11(6)**.

GOLDEN, B. (2004) *Succeeding with open source*. Boston: Addison-Wesley.

GRADY, R.B. and CASWELL, D.L. (1987) *Software Metrics: Establishing a Company-wide Program.* New Jersey: Prentice Hall.

HALLORAN, T.J. and SCHERLIS, W.L. (2002) High Quality and Open Source Software Practices. *in* ICSE 2002, *Proceeding of the 2nd Workshop on Open Source Software Engineering, Orlando, FL, USA*.

HARS, A. and OU, S. (2001) Working for Free?-Motivations of Participating in Open Source Projects. *in Proceedings: 34th Annual Hawaii International Conference on System Science*. pp. 2284–2292.

HAMBLING, B. (1996) *Managing Software Quality.* Mc Graw Hill, p. 4

HENDERSON, J. and J. COOPRIDER, (1990) Dimensions of I/S Planning and Design Aids: A Functional Model of CASE Technology. *Information Systems Research*, **1**, pp. 227–254.

HIRSCHHEIM, R. (1985) *Information System Epistemology: An Historical Perspective*. *in* MUMFORD, E., HIRSCHHEIM, R., FITZGERALD, G. and WOOD-HARPER, T. (eds.) *Research Method in Information Systems.* North-Holland: Elsevier Science Publishers, pp. 9–33.

HIRSCHHEIM, R. and KLEIN, H.K. (1989) Four Paradigms of Information Systems Development. *Communications of the ACM*, **32(10)**, pp. 1199–1216.

HOFFMANN, N. (1999) *Open Source Software* [online]. [cited 8th October 2003]. <http://public.kitware.com/VTK/pdf/oss.pdf>

HOYER, R.W. and HOYER, B.B.Y. (2001) "What is quality?". *Quality Progress,* **7**, pp. 52-62.

IEEE 610 (1991) Standards Coordinating Committee of the IEEE Computer Society. *IEEE Standard Glossary for Software Engineering Terminology*, **IEEE-STD-610.12-1990**, New York.

IEEE 730 (1998) IEEE Standard for Software Quality Assurance Plans. **IEEE Std 730-1998**, New York : IEEE Computer Society.

IEEE 1061 (1998) Standard for a Software Quality Metrics Methodology, revision. *IEEE Standard Department,* **IEEE Std. 1061-1998**, Piscataway, NJ, p. 2.

ISO 8402 (1994) *Quality Management and Quality Assurance Vocabulary.* International Organization for Standardization, Geneva.

ISO 9001:2000 (2000) *Quality management systems – Requirements*. International Organization for Standardization, Geneva.

ISO/IEC 9126-1 (2001) *Software Engineering -- Product Quality -- Part 1: Quality Model*. International Organization for Standardization, Geneva.

ISO/IEC 12207 (2007) *Systems and Software Engineering – Software life cycle processes*. International Organization for Standardization, Geneva.

ISO/IEC 14598-1 (1999) *Software product evaluation—Part 1: General overview*. International Organization for Standardization, Geneva.

ISO/IEC 15504-5 (1998) *Information technology – Process Assessment – Part 5: An exemplar Process Assessment Model*. International Organization for Standardization, Geneva.

IIVARI, J. (1991) A Paradigmatic Analysis of Contemporary Schools of IS Development. *European Journal of Information Systems*, **1(4)**, pp. 249–272.

IIVARI, J., HIRSCHHEIM, R. and KLEIN, H.K. (1998) A Paradigmatic Analysis Contrasting Information Systems Development Approaches and Methodologies. *Information Systems Research*, **9(2)**, pp. 164–193.

IIVARI, J. (2005) An empirical test of the DeLone-McLean model of information system success. *ACM SIGMIS Database*, **36(2)**, pp. 8–27.

JARVENPAA, S.L. (1988) The importance of laboratory experimentation in IS research (technical correspondence). *Communications of the ACM*, **31(12)**, pp. 1502–1504.

JICK, T.D. (1979) Mixing Qualitative and Quantitative Methods: Triangulation in Action. *Administrative Science Quarterly*, **24(4)**, pp. 602–611.

JENSEN, C. and SCACCHI, W. (2005) Modeling Recruitment and Role Migration Processes in OSSD Projects. Institute for Software Research, University of California, [cited 19th April 2007]. <http://www.ics.uci.edu/~wscacchi/Papers/New/Jensen-Scacchi-ProSim05.pdf>

JURAN, J. M. (1979) *"Basic Concepts". in* JURAN, J. M., GRYNA, F. M. (eds.) *Quality Control Handbook.* 3rd ed., New York: McGraw-Hill Book Co.

KANIJ, D. and HENRY, S. (1990) Total quality management: the second industrial revolution. *Total Quality Management*, **1(1)**, pp. 3–12.

KINNULA, A. (2001) *Software process engineering systems: Models and industry cases.* Oulo: University Press, pp. 117.

KIRK, J. and MILLER, M.L. (1986) *Reliability and validity in qualitative research*. Sage.

KITCHENHAM, B. and PFLEEGER, S.L. (1996) "Software Quality: The Elusive Target." *IEEE Software,* **1**, p. 13.

KNOELL, H.D., OTTE, T. and MORETON, R. (2008) Quality Assurance Methods and the Open Source Development Model. *FINAL Information Science: Technical Report and Working Papers*, **18(1)**, Leuphana University, Lüneburg, pp. 95–106.

KOMI-SIRVIÖ, S. (2004) *Development and Evaluation of Software Process Improvement Methods*. VTT Publications 535, Espoo, p. 44.

KORU, G. and TIAN, J. (2004) Defect Handling in Medium and Large Open Source Projects. *IEEE Software,* **4**, pp. 54–61.

KRAEMER, K.L. (eds.) (1991) *The Information Systems Research Challenge: Survey Research Methods*. Boston: Harvard Business School, pp. 299–315.

KRAEMER, K.L. and DUTTON, W.H. (1991) Survey Research in the Study of Management Information Systems. *in* KRAEMER, K.L. (ed.) *The Information Systems Research Challenge: Survey Research Methods*. vol. 3., Cambridge, MA: Harvard Business School Press, pp. 3–58.

KUGLER, H.-J. and MESSNARZ, R. (1994) From the software process to software quality: BOOTSTRAP and ISO9000. *in Proceedings: Software Engineering Conference, December 1994*, **7(9)**, pp. 174–182.

KUVAJA, P., SIMILA, J., KRANIK, L., BICEGO, A., SAUKKONEN, S. and KOCH, G. (1994) *Software Process Assessment & Improvement. The Bootstrap approach*. Blackwell.

LERNER, J. and TRIOLE, J. (2002) Some Simple Economics of Open Source. *Journal of Industrial Economics*, **46(2)**, pp. 125–156.

LUBBE, S. (2003) The Development of a Case Study Methodology in the Information Technology (IT) Field: A Step by Step Approach. *Ubiquity - An ACM IT Magazine and Forum*, **4(27)**, pp. 26.

MACCORMACK, A.D., VERGANTI, R. and IANSITI, M. (2001) Developing product on 'internet time': The anatomy of a flexible development process. *Management Science*, **47 (1)**, pp. 133–150.

MALHOTRA, M.K. and GROVER, V. (1998) An assessment of survey research in POM: from constructs to theory. *Journal of Operations Management*, **16(4)**, pp. 407–425.

MASSEY, B. (2003) Why OSS folks think SE folks are clue-impaired. *in* ICSE 2003, *Proceedings of the 3rd Workshop on Open Source Software Engineering*, pp. 91–97.

MCCALL, J.A., RICHARDS, P.K. and WALTERS, G.F. (1977) *Factors in software quality*. RADC-TR-77-369. Griffiss Air Force Base, NY, Rome Air Development Center, Air Force Systems Command.

MICHLMAYR, M. (2005) Software Process Maturity and the Success of Free Software Projects. *in* ZIELIŃSKI, K. and SZMUC, T. (eds.) *Software engineering: evolution and emerging technologies*. Amsterdam: IOS Press, pp. 3–14.

MICHLMAYR, M. (2005a) Quality Improvement in Volunteer Free Software Projects: Exploring the Impact of Release Management. *in Proceedings of the First International Conference on Open Source Systems, 11-15 July 2005*, *Genova,* pp. 309–310.

MICHLMAYR, M. and HILL, B.H. (2003) Quality and the Reliance on Individuals in Free Software Projects. *in* ICSE 2003, *Proceedings of the 3rd Workshop on Open Source Software Engineering,* pp. 105–109.

MICHLMAYR, M., HUNT, F. and PROBERT, D. (2005) Quality Practices and Problems in Free Software Projects. *in Proceedings of the First International Conference on Open Source Software Systems, 11-15 July 2005, Genova,* pp. 24–28.

MICHLMAYR, M. (2007) Quality Improvement in Volunteer Free and Open Source Software Projects - Exploring the Impact of Release Management. PhD thesis, King's College, University of Cambridge.

MILES, M.B., HUBERMAN, A.M. (1994) *Qualitative Data Analysis: A Sourcebook of New Methods*. London: Sage Publications.

MILLER, B., KOSKI, D., LEE, C., MAGANTY, V., MURTHY, R., NATARAJAN, A. and STEIDL, J., (1995) *Fuzz revisited: a re-examination of the reliability of UNIX utilities and services.* [cited 27th March 2005]. <http://www.cs.wisc.edu/~bart/fuzz/fuzz.html>.

MOCKUS, A., FIELDING, R.T. and HERBSLEB, J.D. (2002) Two Case Studies of Open Source Software Development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology*, **11(3),** pp. 1–38.

MYERS, M.D. (1997) Qualitative Research in Information Systems. *MIS Quarterly*, **21(2)**, pp. 241–242.

NAVICA (2007) *The open source maturity model (OSMM)* [online]. Navica Inc., [cited 10th October 2007]. <http://www.navicasoft.com/pages/osmmoverview.htm>.

NEWSTED, P.R., CHIN, W., NGWENYAMA, O. and LEE, A. (1996) Resolved: Surveys have Outlived their Usefulness in IS Research. *in Proceedings of the 1996 International Conference on Information Systems, 17 December 1997, Cleveland, Ohio.*

NEWSTED, P.R., HUFF, S. and MUNRO, M. (1998) Survey Research in Information Systems, MISQ Discovery, December.

NOWAK, P. (2003) *The Powerful Economic Underpinnings of OSS* [online]. LinuxWorld, [cited 5th December 2003]. <http://www.linuxworld.com/story/34293.htm>.

OPENPROJECTS (2005) *The Computer Software Definition* [online]. Openprojects, [cited 5th August 2005]. <http://www.openprojects.org/software-definition.htm>

OSI (2005) *The Open Source Definition* [online]. Open Source Initiative, [cited 18th March 2005]. <http://www.opensource.org/docs/definition.php>.

OSI (2007) *Open Source Case for Business: Advocacy* [online]. Open Source Initiative, [cited 24th April 2007]. <http://opensource.mirroring.de/advocacy/case_for_business.php>.

OSQ (2007) *Open source quality (OSQ) project* [online]. OSQ, [cited 10th October 2007]. <http://osq.cs.berkeley.edu/>.

OTTE, T., MORETON, R. and KNOELL, H.D. (2008a) Applied Quality Assurance Methods under the Open Source Development Model. *in Proceedings 32nd Annual IEEE International Computer Software and Application Conference 2008, 28.7.-1.8.2008, Turku, Finland,* pp. 1247–1252.

OTTE, T., MORETON, R. and KNOELL, H.D. (2008b) Development of a Quality Assurance Framework for the Open Source Development Model. *in Proceedings Annual IEEE 3rd International Conference on Software Engineering Advances ICSEA 2008, October 26-31, 2008, Sliema, Malta,* pp. 123–131.

PAULHUS, D.L. (1991) Measurement and Control of Response Bias. *in* ROBINSON, P., SHAVER, P.R. and WRIGHTSMAN, L.S., (eds.), *Measures of Personality and Social Psychological Attitudes*. San Diego, C.A.: Academic Press, pp. 17–59.

PAULK, M.C., CURTIS, B., CHRISSIS, M.B. and WEBER, C.V. (1993) *Capability Maturity Model for Software, Version 1.1 Technical Report.* CMU/SEI-93-TR-024, pp. 1–14.

PAULK, M.C. (1994) *A Comparison of ISO 9001 and the Capability Maturity Model for Software.* Technical Report, CMU/SEI-94-TR-12, pp. 9–10.

PAULK, M.C., WEBER, C.V, CURTIS, B. and CHRISSIS, M.B. (1994b) *Capability Maturity Model - The Guidelines for Improving the Software Process.* Addison-Wesley, p. 441.

PINSONNEAULT, A. and KRAEMER, K.L. (1993) Survey research methodology in management information systems: An assessment. *Journal of Management Information Systems*, **10(2)**, pp. 75.

PEELING, N., and SATCHELL, J. (2001) Analysis of the Impact of Open Source Software [online]. QinetiQ Ltd. QINETIQ/KI/SEB/CR010223, [cited 3rd April 2007]. <http://www.govtalk.gov.uk/documents/QinetiQ_OSS_rep.pdf>.

PERRY, W.E. (1987) *Effective Methods for EDP Quality Assurance*. 2nd ed., Prentice Hall.

PERRY, W.E. (1991) *Quality Assurance for Information Systems: Methods, Tools, and Techniques.* Wellesley, MA: QED Technical Publishing Group.

PERRY, D., STAUDENMAYER, P. and VOTTA, L. (1994) People, organizations, and process improvement. *IEEE Software*, **11(4)**, pp. 36–45.

PETTIGREW, A.M. (1985) *Contextualist Research and the Study of Organisational Change Processes*. *in* MUMFORD, E., HIRSCHHEIM, R., FITZGERALD, G. and WOOD-HARPER, T. (eds.) *Research Method in Information Systems.* North-Holland: Elsevier Science Publishers, pp. 53–72.

PFLEEGER, S.L. (2001) *Software engineering: Theory and practice*. 2nd edition, Upper Saddle River, N.J.: Prentice Hall.

PIRSIG, R.M. (1974) *Zen and the Art of Motorcycle Maintenance*. New York: William Morrow & Co, pp. 185–213.

PORST, R. (1998) Im Vorfeld der Befragung. *ZUMA-Arbeitsbericht 98/02*, Mannheim.

PORTER, A. and VOTTA, L. (1995) Comparing detection methods for software requirements inspections: a replication using professional subjects. *Empirical Software Engineering: An International Journal*, **3(4)**, pp. 355–379.

PORTER, A, YILMAZ, C., MENON, A.M., KRISHNA, A.S., SCHMIDT, D.C. and GOKHALE, A. (2006) Techniques and processes for improving the quality and performance of open-source software. *Software Process: Improvement and Practice - Special Issue on Free/Open Source Software Processes*, **11(2)**, pp. 163–176.

PRESSMAN, R.S. (1992) *Software Engineering - A Practitioner's Approach*. 3rd Edition, McGraw-Hill.

PUTNAM, L.H. and MYERS, W. (1992) *Measures for Excellence: Reliable Software on Time, within Budget*. P.T.R. Prentice Hall, Inc., Englewood Cliffs, N.J.

QSOS (2006) *Method for Qualification and Selection of Open Source software (QSOS) version 1.6* [online]. QSOS Project, [cited 10th December 2007]. <http://www.qsos.org/download/qsos-1.6-en.pdf>.

QUALISPO (2007) *Quality Platform for Open Source Software (QUALISPO)* [online]. Qualispo, [cited 10th October 2007]. <http://www.qualipso.org/index.php>.

QUALOSS (2007) *Quality in Open Source Software – Related Projects* [online]. Qualoss, [cited 9th October 2007]. <http://www.qualoss.org/related-projects>.

RADICE, R.A. (2002) *High quality low cost software inspections*. Andover, MA: Paradoxicon Publishing.

RALSTON, A. and REILLY, E. (1993) *Encyclopaedia of Computer Science.* 3rd ed., New York: Van Nostrand Reinhold Co., p. 226.

RAPOPORT, R.N. (1970) Three Dilemmas in Action Research. *Human Relations*, **23(4)**, pp. 499–513.

RASTERS, G. (2004) *Communication and Collaboration in Virtual Teams*. The Netherlands: Ipskamp.

RAYMOND, E.S. (1999) Linux and open-source success. *IEEE Software*, **16(1)**, pp. 85–89.

RAYMOND, E.S. (2001) *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary.* revised ed., O'Reilly.

REIFER, D.J. (1985) *State of the Art in Software Quality Management*. Reifer Consultants, Torrance: California, p.20.

REIFER, D.J. (2004) Industry Software Cost, Quality and Productivity Benchmarks. *The DoD Software Tech News*, **7(2)**.

ROMBACH, D. (2003) *Software Engineering* [online]. Universität Kaiserslautern, [cited 13th April 2005]. <http://wwwagse.informatik.uni-kl.de/teaching/se2/ss2004/unterlagen/>.

ROSENBERG, D.K. (2000) *Open Source: The Unauthorized White Papers.* Foster City: M&T Books.

ROTHERMEL, G. and HARROLD, M. (1996) Analyzing regression test selection techniques. *IEEE Transaction on Software Engineering*, **22(8)**, pp. 529–551.

ROTHFUSS, G.J. (2002) *A Framework for Open Source Projects* [online]. [cited 17th October 2007]. <http://opensource.mit.edu/papers/rotfuss.pdf>

SATPATHY, M., HARRISON, R., SNOOK, C. and BUTLER, M. (2000) A Generic Model for Assessing Process Quality. *in* DUMKE, R., ABRAN, A. (eds.) *New Approaches in Software Measurement, 10th International Workshop, IWSM 2000, Berlin: Germany,*

October 4-6, Proceedings. Lecture Notes in Computer Science 2006. Berlin, Heidelberg: Springer-Verlag, pp. 95–110.

SCHMIDT, D.C. and PORTER, A. (2001) Leveraging open-source communities to improve the quality & performance of open-source software. *in Proceedings of the 1st Workshop on Open Source Software Engineering,* ICSE, Toronto: Canada.

SCHULMEYER, G.G. and MCMANUS, J.I. (1999) *Handbook of Software Quality Assurance.* 3rd ed., Upper Saddle River, NJ: Prentice-Hall.

SEDDON, P.B. (1997) A Respecification and Extension of the DeLone and McLean model of IS Success. *Information Systems Research*, **8(3)**, pp. 240–253.

SEDDON, P.B., STAPLES, S., PATNAYAKUNI, R., and BOWTELL, M. (1999) Dimensions of information systems success. *Communications of the Association for Information Systems*, **2(20)**, pp. 1–61.

SHAIKH, S. and CERONE, A. (2007) Towards a Quality Model for Open Source Software. *in OpenCert 2007, 1st International Workshop on Foundations and Techniques for Open Source Software Certification, 31 March 2007, Braga: Portugal*.

SHEARD, S.A. (1997) The Frameworks Quagmire. *Crosstalk – The Journal of Defense Software Engineering*, **9**.

SHEARD, S. (2001) Evolution of the Frameworks Quagmire. *IEEE Computer*, **34(7)**, pp. 96–98.

SIMSEK, Z., VEIGA, J.F. (2000) The Electronic Survey Technique: An Integration and Assessment. *Organizational Research Methods*, **3(1)**, pp. 93–115.

SINGH, R. (1998) International Standard ISO/IEC 12207 Software Life Cycle Processes [online]. Federal Aviation Administration, Washington, DC, [cited 12th October 2007]. <http://www.abelia.com/docs/12207cpt.pdf>.

STAMELOS, I., ANGELIS, L., OIKONOMOU, A. and BLERIS, G.L. (2002) Code quality analysis in open source software development. *Information Systems Journal*, **12(1)**, pp. 43–60.

STRAUB, D.W. (1989) Validating Instruments in MIS Research. *MIS Quarterly*, **13(2)**, pp. 147–169.

STRAUB, D.W., GEFEN, D. and BOUDREAU, M.C. (2004) Validation Guidelines for IS Positivist Research. *Communications of the Association for Information Systems*, **14**, pp. 380–426.

STRAUB, D.W., GEFEN, D. and BOUDREAU, M.C. (2005) Quantitative Research. *in* AVISON, D. and PRIES-HEJE, J. (eds.) *Research in Information Systems: A Handbook for Research Supervisors and Their Students.* Amsterdam: Elsevier, pp. 221–238.

SUDMAN, S. and BRADBURN, N.M. (1982) *Asking Questions*. San Francisco: Jossey-Bass.

SUSMAN, G. and EVERED, R. (1978) An Assessment of The Scientific Merits of Action Research. *Administrative Science Quarterly*, **23(4)**, pp. 582–603.

SQM (1993) A Review of the Managing Software Quality in the 90s. Conference by Elliot Marley. *SQM*, **18 Spring**, pp. 24–28.

TINGEY, M.O. (1997) *Comparing ISO 9000, Malcolm Baldrige, and the SEI CMM for software – a reference and selection guide*. Upper Saddle River, NJ: Prentice and Hall PTR.

TUMA, D. (2005) Open Source Software: Opportunities and Challenges. *Cross Talk - The Journal of Defense Software Engineering*, **18(1)**, pp. 6–10.

VIDICH, A.J. and SHAPIRO, G. (1955) A Comparison of Participant-Observation and Survey Data. *American Sociological Review*, **20**, pp. 28–33.

VILLA, L. (2003) Large free software projects and Bugzilla. *in Proceedings of the Linux Symposium, 23-26 July 2003, Ottawa, Canada.*

VITALARI, N.P. (1985) The Need for Longitudinal Designs in the Study of Computing Environments. *in* MUMFORD, E., HIRSCHHEIM, R., FITZGERALD, G. and WOOD-HARPER, T. (eds.) *Research Method in Information Systems.* North-Holland: Elsevier Science Publishers, pp. 243–265.

VINSON, N.G. and SINGER, J. (2001) Getting to the Source of Ethical Issues. *Empirical Software Engineering*, **6(4),** pp. 293–297.

VIXIE, P., (1999) Software Engineering. *in* DIBONA, C., OCKMANN, S., STONE, M. (1999) *OpenSources - Voices from the Open Source Revolution.* O'Reilly, pp. 91–100.

WALSHAM, G. (1993) Interpreting Information Systems in Organizations. Wiley, Chichester.

WALSHAM, G. (1995a) Interpretive case studies in IS research: nature and method. *European Journal of Information Systems*, **4**, pp. 74–81.

WALSHAM, G. (1995b) The Emergence of Interpretivism in IS Research. *Information Systems Research,* **6(4)**, pp. 376–394.

WALTON, M. (1986) *The Deming management method*. New York, NY: Perigee.

WANG, Y. and KING, G. (2000) *Software Engineering Processes – Principles and Applications*. Boca Raton London New York: CRC Press.

WHEELER, D. A. (2004) Open Source Software (OSS) and Security. *in Proceeding of the Third Annual Open Source in Government Conference, 15-17 March 2004, George Washington University, D.C.*

WICHMANN, T. and SPILLER, D. (2002) *FLOSS Final Report - Part 3 – Free/Libre Open Source Software: Survey and Study*. Berlecon Research, Berlin, [cited 23rd April 2007]. <http://www.berlecon.de/studien/downloads/200207FLOSS_Basics.pdf>.

WIELAND, T. (2004) Stärken und Schwächen freier und Open-Source-Software im Unternehmen. *in* GEHRING, R. A. and LUTTERBECK, B. (eds.) *Open Source Jahrbuch 2004.* Berlin: Lehmanns Media, pp. 107–119.

WONG, B. (2006) The Different Views of Software Quality. *in* DUGGAN, E.W. and REICHGELT, H. (eds.) *Measuring Information Systems Delivery Quality*. Hershey: Idea Group Publishing, pp. 55–88.

WORKING GROUP ON LIBRE SOFTWARE (2000) *Free software/open source: Information society opportunities for Europe*. Technical report, Information Society Directorate General of the European Commission, April 2000.

YIN, R.K. (1993) Applications of Case Study Research. *in* BICKMAN, L., ROG, D. J. (eds.) *Applied Social Research Methods Series*. London: Sage Publications.

YIN, R.K. (2002) Case Study Research, Design and Methods. 3rd ed. Newbury Park: Sage Publications.

YOUNESSI, H. (2002) *Object-Oriented Defect Management of Software*. 1st ed., Prentice Hall.

ZHANG, X. and PHAM, H., (2000) An analysis of factors affecting software reliability. *Journal of Systems and Software,* **50(1)**, pp. 43–56.

ZHAO, L. and ELBAUM, S. (2000) A survey on software quality related activities in open source. *ACM SIGSOFT Software Engineering Notes*, **25(3)**, pp. 54–57.

ZHAO, L. and ELBAUM, S. (2003) Quality assurance under the open source development model. *The Journal of Systems and Software,* **66**, pp. 65–75.

Part 4 - Appendix

List of Figures

List of Tables

A. Survey Research Approach

A.1 Survey Rationale

Table A.1. Survey Research Targets

ID	Topic	Target	Rationale
General Issues			
G1	Project title	Project name	Identification of project
G2	Project type	Topic of application	The topic allows a clustering and classification of project
G3	Project size in LOC	Identification of project size	The project size indicates the project complexity
G4	Development language	Definition of development languages	The development language shows an overall picture and may allow the identification of dependencies of the applied languages.
G5	Project developer team size	Identification of test and verification possibilities due to other developer	The determination of the number of developer shows the project size and the potential for collaborative processes, such as code reviews
G6	Project community size	Identification of test and verification possibilities due to users	The determination of the number of user shows the project community size and the potential for user testing and indicates the project attraction.
G7	Market availability	Determination of project maturity level	The market availability indicates the project maturity and shows indirectly the project success.
G8	Project maturity	Maturity level of project	The determination of the project maturity level indicates its stability and may show how long projects need to achieve a productive status.
Personal Issues			
S1	Main project role	Identification of main project role	The participant's main role provides evidence about the target group give an indicator about the response quality in terms of qualified response.
S2	Further project roles	Identification of role model and shared capabilities	The determination shows if a multifaceted role model exists.
S3	Development experience	Determination of development experience	The relationship of experience to success may provide evidence that OSS projects are not only successful due to highly experiences participants, but because of the approach.
S4	Project Participation	Project participation	The developer contribution and participation indicates the time spent on the projects.
S5	Developer / user satisfaction	Personal satisfaction	The level of motivation may show if developers are consumers of their product or if a strong commercial focus exists.
Processes and Methods			
P1	Proportion of activities	Proportion of project design, coding and testing activities	The proportion of development activities in relation to project size shows the correlation and the average effort spent.
P2	Release frequency	Identification of release frequency	The release frequency shows the level of activity and indicates how frequent changes are delivered, which needs to be set in correlation of the project size and maturity.
P3	Proportion of changes from release to release	Determines the level of code changes	The level of code changes shows the project activity and is an indirect project success factor.
P4	Time versus feature based release strategy	Identifies the release strategy	The release strategy is a general criterion and may indicate the tendencies in correlation to the project size and the project success.
P5	Reusability of code	Determination of average proportion of reused code	The proportion of reused code may show if projects that largely reuse code have a reduced testing effort or could increase their code quality.
P6	Modularity	Development code modularity	The level of modularity may indicate the level of code quality and show if an early consideration during design activities, project size or maturity exists.
P7	Level of abandoned code	Determination of level of abandonment code	The level of abandoned code indicates the project complexity, the manageability and its maintainability.
P8	Quality problems during maintenance	Examination of project complexity issues and side effects during maintenance	The degree of quality issues may indicate the project complexity (in correlation with the level of abandoned code)

Testing			
T1	Testing proportions	Determines the proportion of testing activities	The proportion of testing activities shows how much effort projects spent into testing
T2	Structured testing	Are defined processes established	The analysis may confirm if a structured testing approach is applied and what correlation to project size and maturity exists.
T3	How much code is tested by user	Explores the degree of user testing	The level of user testing determines how much projects leverages their communities.
T4	Modification or user suggestions	Analysis the user involvement	The level of user suggestions may explain if projects listen to their "customers" and indicates the degree of user driven development direction.
T5	Testing by users	Determines the efficiency of user testing activities	The degree of user testing shows the testing efficiency and indicates the user contribution to achieve high quality
T6	Code reviews/inspections by other developer	Examination of unsolicited code reviews by developer	The examination shows if code reviews are unsolicited performed.
T7	Proportion of reviews / inspections	Determination of the level of reviews and inspections	The analysis shows the degree of code reviews or inspections and its correlation regarding mature projects.
T8	Peer reviews	Exploration of additional formal review method	The determination shows the degree of peer review approaches and its correlation to mature projects.
T9	Walkthroughs or code readings	Analysis of informal review methods	The analysis shows the level of informal review approaches.
T10	Quality control before commit	Examines the degree of quality control	The study indicates how strict quality control is applied and what correlation to mature projects exists.
Defect Handling			
T11	Introduction of defect handling	Determines the point of time	The examination indicates the point of time when defect-handling processes are introduced.
T12	Scope of defect handling	Scope of defect handling	The analysis shows the scope of defect handling processes.
T13	Defect reporting quality	Quality of defect handling processes	The examination shows the degree of reporting quality and its efficiency.
T14	Defect handling process	Defect handling process responsibilities	The analysis examines the organisational setup and handling within the project.
T15	Defect handling time	Speed of defect handling process	The defect handling time indicates the process effectivity and the level of project activity.
T16	Security critical defect handling	Determines the prioritization for security critical defect handling	The analysis shows the degree how projects handle security critical defects and ensures security critical patches in time.
T17	Defect follow-up	Examines the defect follow-up handling	The analysis shows the level of status tracking and the process efficiency.
Organisational Issues			
C1	On-boarding	Examines the quality of the on-boarding process	The analysis indicates how effective knowledge transfer is performed.
C2	On-boarding time to productivity	Determines the time spent until new resources are productive.	The degree explains the efficiency of knowledge transfer processes and its correlation to project size.
C3	Communication	Analysis the quality of project communication	The analysis determines the effectivity of communication and coordination processes and shows if a correlation to the project size exists.
D1	Process and project documentation	Analysis of documentation quality	The examination shows the degree of documentation quality and may indicate the manageability as well as the on-boarding quality.
D2	Development documentation	Explores the developer documentation quality	The investigation shows the quality of the development process, which indirectly affects the product quality.
D3	Product documentation	User documentation quality	The study measures the product quality and indicates the level of user satisfaction.
I1	Tool usage	Identification of tool usage regarding Web Portals, Source Code Control, Code Viewers, Automatic Build Tools, Mailing Lists, Bug Tracking Tools, Test Support Tools and Instant Messaging	The examination shows the degree of tool usages and its correlation to project size, maturity and success.
Quality Assurance Tasks			

Q1	Execution of QA practices	Determination if Quality Assurance are applied	The analysis shows if QA processes are applied and what correlation to the project size exist.
Q2	Results checked by QA	Exploration if Quality Assurance check if results deviate from plan	The study shows the effectiveness of QA, which may differ in relation to the project size.
Q3	QA action performed	Analysis if QA triggers activities to influence the development processes	The examination explores the improvement activities and may show differences in relation to the project maturity.

A.2 Survey Questionnaire

01 General Data

G1 *What is the name of the project you contribute the most to?

_____

G2 *What is the application type you develop?

Please choose..

○ Communications
○ Database
○ Education
○ ERP/CRM/Financial
○ Front-Ends
○ Games/Entertainment
○ Internet
○ Multimedia
○ Networking
○ Office/Business
○ Security
○ Software Development
○ Other: _____

G3 *Please specify appr. the number of Lines of Code (LOC) within the project

Please choose..

○ <1.000
○ 1.000-10.000
○ 10.000-100.000
○ >100.000
○ I don't know

G4 *Which development language do you use?

Please choose..

○ JAVA
○ C++
○ C
○ PHP
○ Perl
○ Python
○ JavaScript
○ Unix shell
○ Delphi
○ Visual Basic
○ Assembly
○ PL/SQL

○ JSP
○ Other: _____

G5 *How many developers contribute to the project?
Please choose..
○ 1
○ 2-5
○ 6-10
○ 11-20
○ >20

G6 *How many users participate in the project?
Please choose..
○ <10
○ 10-50
○ 51-100
○ >100

G7 *How many years has the product been on the market?
Please choose..
○ <0.5 year
○ 0.5-1 year
○ 1-2 years
○ 2-4 years
○ >4 years

G8 *What is the current release status of the product?
Please choose..
○ Inactive
○ Planning
○ Pre-Alpha
○ Alpha
○ Beta
○ Productive

02 Statistics

S1 *What best describes your main role on the project?
Please choose..
○ No specific role
○ User
○ Advisor
○ Engineering/Design
○ Developer
○ Tester
○ Translator
○ Project Manager

S2 If you have more than one role within the project, please specify what they are
☐ User
☐ Advisor
☐ Engineering/Design
☐ Developer
☐ Core Developer
☐ Contributor
☐ Release Manager
☐ Maintainer
☐ Tester
☐ Translator
☐ Project Manager
☐ Other: _____

S3 *How many years of experience do you have in software development?
Please choose..
○ <1 year
○ 1-2 years
○ 2-5 years
○ >5 years

[Only answer this if the following conditions are met:]
-to question 'S1', you answered 'Advisor' OR 'Engineering/Design' OR 'Developer' OR 'Tester' OR 'Project Manager' OR 'No specific role'

S4 *What is your level of participation?
Please choose..
○ full-time
○ part-time
○ seldom
○ passive

S5 *Why do you participate in this project?
Please choose..
○ personal needs
○ company needs
○ community needs
○ Other: _____

03 Processes and Methods

P1 *Please specify what percentage the following phases have - compared to the whole project time (please consider that the sum of all phases needs to be 100%)
Please choose..

	0%	10	20	30	40	50	60	70	80	90	100
Design Phase	○	○	○	○	○	○	○	○	○	○	○
Coding	○	○	○	○	○	○	○	○	○	○	○
Testing	○	○	○	○	○	○	○	○	○	○	○

P2 *How often do you release a new version?
Please choose..
○ 1 - once a week
○ 2 - every two weeks
○ 3 - once a month
○ 4 - once a quarter
○ 5 - once half year
○ 6 - above

P3 *What proportion of the code has changed between major releases?
Please choose..
○ 0-10%
○ 10-20%
○ 20-30%
○ 30-50%
○ >50%

P4 *The release strategy in the project is driven ...
Please choose..
○ 1 - time-based (fixed milestones)
○ 2 – feature based (defined scope)
○ 3 - I don't know
○ Other: _____

[Only answer this if the following conditions are met:]
-to question 'S1', you answered 'Advisor' OR 'Engineering/Design' OR 'Developer' OR 'Tester' OR 'Project Manager' OR 'No specific role'

P5 *What percentage of the source code were re-used in this project?
Please choose..
○ 0-5%
○ 5-10%
○ 10-20%
○ 20-40%
○ 40-60%
○ >60%

[Only answer this if the following conditions are met:]
-to question 'S1', you answered 'Advisor' OR 'Engineering/Design' OR 'Developer' OR 'Tester' OR 'Project Manager' OR 'No specific role'

P6 *How is modularity considered within the development?
Please choose..
○ 1 - already defined in the design phase
○ 2 - considered with start of the development phase
○ 3 - reconsidered during development
○ 4 - no modular development

[Only answer this if the following conditions are met:]
-to question 'S1', you answered 'Advisor' OR 'Engineering/Design' OR 'Developer' OR 'Tester' OR 'Project Manager' OR 'No specific role'

P7 *How much of the code do you estimate as abandoned?

Please choose..

○ 0-5%
○ 5-10%
○ 10-20%
○ 20-30%
○ 30-40%
○ >40%

[Only answer this if the following conditions are met:]
-to question 'S1', you answered 'Advisor' OR 'Engineering/Design' OR 'Developer' OR 'Tester' OR 'Project Manager' OR 'No specific role'

P8 *Please assess how problems due to code complexity or side effects affect your development process?

Please choose..

○ 1 - no problems occurred
○ 2 - minor problems exist
○ 3 - major problems exist
○ 4 - cause huge problems

04 Testing

T1 *What is the proportion of the testing time compared to the whole development time?

Please choose...

○ 0-20%
○ 20-40%
○ 40-60%
○ 60-80%
○ 80-100%

T2 *What is your testing approach?

Please choose..

○ 1 - I don't know
○ 2 - I had no testing plan
○ 3 - I executed testing according experience
○ 4 - some test scripts available but no planning
○ 5 - defined test cases with testing phases

T3 *How much of the code is tested by the user?

Please choose..

○ 0-20%
○ 20-40%
○ 40-60%
○ 60-80%
○ 80-100%

[Only answer this if the following conditions are met:]
-to question 'S1', you answered 'Advisor' OR 'Engineering/Design' OR 'Developer' OR 'Tester' OR 'Project Manager' OR 'No specific role'

T4 *How do you consider user modifications or comments?

Please choose..

○ 1 - bring the design forward
○ 2 - sometimes useful
○ 3 - not very efficient
○ 4 - do not fit into the design

[Only answer this if the following conditions are met:]
-to question 'S1', you answered 'Advisor' OR 'Engineering/Design' OR 'Developer' OR 'Tester' OR 'Project Manager' OR 'No specific role'

T5 *How do you assess the defects found by the user?

Please choose..

○ 1 - user could not help much
○ 2 - supported me but I would have found the bugs also on my own
○ 3 - user found major defects or hard bugs
○ Other: _____

[Only answer this if the following conditions are met:]
-to question 'S1', you answered 'Advisor' OR 'Engineering/Design' OR 'Developer' OR 'Tester' OR 'Project Manager' OR 'No specific role'

T6 *Do you directly ask other developers to formally review the code?

Please choose..

○ 1 - code reviews are frequently conducted
○ 2 - only critical or important code is reviewed
○ 3 - code reviews are very seldom
○ 4 - no reviews conducted

○ 5 - I don't know Please choose..
○ Other: _____

[Only answer this if the following conditions are met:]
-to question 'S1', you answered 'Advisor' OR 'Engineering/Design' OR 'Developer' OR 'Tester' OR 'Project Manager' OR 'No specific role'

T7 *How much of the code is reviewed or inspected?

Please choose..
○ 0-20%
○ 20-40%
○ 40-60%
○ 60-80%
○ 80-100%

[Only answer this if the following conditions are met:]
-to question 'S1', you answered 'Advisor' OR 'Engineering/Design' OR 'Developer' OR 'Tester' OR 'Project Manager' OR 'No specific role'

T8 *Do you use peer reviews to improve code quality?

Please choose..
○ 1 - peer reviews are elementary part of our testing approach
○ 2 - they are conducted on request
○ 3 - for critical code only
○ 4 - not applied
○ 5 - I don't know
○ Other: _____

[Only answer this if the following conditions are met:]
-to question 'S1', you answered 'Advisor' OR 'Engineering/Design' OR 'Developer' OR 'Tester' OR 'Project Manager' OR 'No specific role'

T9 *How are informal code reviews, such as walkthroughs or code readings performed?

Please choose..
○ 1 - walkthroughs or code readings are standard procedure
○ 2 - is used sometimes
○ 3 - only in special cases
○ 4 - not used at all
○ 5 - I don't know
○ Other: _____

[Only answer this if the following conditions are met:]
-to question 'S1', you answered 'Advisor' OR 'Engineering/Design' OR 'Developer' OR 'Tester' OR 'Project Manager' OR 'No specific role'

T10 *How do you perform quality control before the source code is committed to the repository?

Please choose..
○ 1 - inappropriate style or structured code is rejected
○ 2 - reworked by the lead developer
○ 3 - integrated with annotations
○ 4 - accepted without further actions
○ 5 - I don't know
○ Other: _____

05 Defect handling

T11 *The defect handling process is introduced...
Please choose..
○ 1 - directly from project start
○ 2 - with start of the design phase
○ 3 - with start of development/coding phase
○ 4 - with start of testing phase
○ 5 - in the post-release phase
○ Other: _____

T12 Please select all appropriate topics what kind of defects you record
☐ 1 - Requirements specification
☐ 2 - Design documentation
☐ 3 - Source Code
☐ 4 - User Documentation
☐ Other: _____

[Only answer this if the following conditions are met:]
-to question 'S1', you answered 'No specific role' OR 'Advisor' OR 'Engineering/Design' OR 'Developer' OR 'Tester' OR 'Project Manager'
T13 *How do you assess the defect reporting effectiveness?
Defect reports contain...
○ 1 - often detailed information with priority and majority
○ 2 - often useful bug descriptions with environmental information
○ 3 - often only short bug description
○ 4 - often unstructured information
○ 5 - often no useful information at all

T14 *How do you manage the defect handling process in the project?
Please choose..
○ 1 - everyone is responsible
○ 2 - defects are assigned to developer
○ 3 - clustering and prioritization with task assignment
○ 4 - no defect handling management

T15 *How do you rate the average defect handling response time?
Please choose..
○ 1 - defects are solved immediately
○ 2 - defects are solved within 1-2 days
○ 3 - defects are solved within a week
○ 4 - takes longer

[Only answer this if the following conditions are met:]
-to question 'S1', you answered 'No specific role' OR 'Advisor' OR 'Engineering/Design' OR 'Developer' OR 'Tester' OR 'Project Manager'
T16 *How do you ensure that security critical defects are solved in time?
Please choose..
○ 1 - we can hardly determine the criticality
○ 2 - there is no special treatment of critical defects
○ 3 - are solved within the whole team
○ 4 - defined escalation procedure with assigned responsibilities exists
○ Other: _____

T17 *How do you track the status of your defects?

Please choose..

- ○ 1 - defects are always classified (e.g. new, old, fixed, ...)
- ○ 2 - defects are sometimes classified
- ○ 3 - no classification exists
- ○ Other: _____

06 Organisation

C1 *A clear process or documentation exist to onboard new volunteers
Please choose.. ○ Yes ○ Uncertain ○ No

[Only answer this if the following conditions are met:]
-to question 'S1', you answered 'No specific role' OR 'Advisor' OR 'Engineering/Design' OR 'Developer' OR 'Tester' OR 'Project Manager'

C2 *How many weeks do you consider for new volunteers to contribute effectively to the project?
Please choose..
○ 1 - less than three days
○ 2 - about one week
○ 3 - 1-2 weeks
○ 4 - 2-3 weeks
○ 5 - 4 weeks and more

C3 *The feedback between developers and users is direct and efficient
Please assess..
○ 1 Strongly Agree
○ 2 Agree
○ 3 Uncertain
○ 4 Disagree
○ 5 Strongly Disagree

[Only answer this if the following conditions are met:]
-to question 'S1', you answered 'No specific role' OR 'Advisor' OR 'Engineering/Design' OR 'Developer' OR 'Tester' OR 'Project Manager'

D1 *How do you assess the documentation of the internal processes?
Please choose..
○ 1 - detailed definition of processes, responsibilities and guidelines exist
○ 2 - some major project guidelines are available
○ 3 - only minor project descriptions exist
○ 4 - we manage the work mostly without process descriptions
○ 5 - processes are common and do not need to be documented
○ Other: _____

[Only answer this if the following conditions are met:]
-to question 'S1', you answered 'No specific role' OR 'Advisor' OR 'Engineering/Design' OR 'Developer' OR 'Tester' OR 'Project Manager'

D2 *Please rate the documentation that describes or supports the development process
Please choose..
○ 1 - coding styles and development guidelines widely available
○ 2 - minor guidelines exist
○ 3 - no guidelines needed as developers are experienced enough
○ 4 - I don't know
○ Other: _____

D3 *How do you assess the end user documentation?
Please choose..
○ 1 - detailed and comprehensive
○ 2 - only important parts documented
○ 3 - draft available but incomplete
○ 4 - not available
○ Other: _____

I1 *Please let us know if you use collaboration tools and if yes which ones

Web Portal	○ Yes	○ Uncertain	○ No
Source Code Control	○ Yes	○ Uncertain	○ No
Code Viewers	○ Yes	○ Uncertain	○ No
Automatic build	○ Yes	○ Uncertain	○ No
Mailing List	○ Yes	○ Uncertain	○ No
Bug tracking Tools	○ Yes	○ Uncertain	○ No
Test support tools	○ Yes	○ Uncertain	○ No
Instant Messaging	○ Yes	○ Uncertain	○ No

[Only answer this if the following conditions are met:]
-to question 'I1', you answered '(Web Portal) : Yes'

I101a Which Web Portal do you use?

[Only answer this if the following conditions are met:]
-to question 'I1', you answered '(Source Code Control) : Yes'

I102a Which Source Code Control Tool do you use? _____

[Only answer this if the following conditions are met:]
-to question 'I1', you answered '(Code Viewers) : Yes'

I103a Which Source Code Viewer do you use? _____

[Only answer this if the following conditions are met:]
-to question 'I1', you answered '(Automatic build) : Yes'

I104a Which Automatic Build Tool do you use? _____

[Only answer this if the following conditions are met:]
-to question 'I1', you answered '(Mailing List) : Yes'

I105a Which Mailing List Tool do you use? _____

[Only answer this if the following conditions are met:]
-to question 'I1', you answered '(Bug tracking Tools) : Yes'

I106a Which Bug Tracking Tool do you use? _____

[Only answer this if the following conditions are met:]
-to question 'I1', you answered '(Test support tools) : Yes'

I107a Which Test Support Tool do you use? _____

[Only answer this if the following conditions are met:]
-to question 'I1', you answered '(Instant Messaging) : Yes'

I108a Which Instant Messaging Tool do you use? _____

I200 Do you use other tools which are not listed?
Please choose.. ○ Yes ○ No

[Only answer this if the following conditions are met:]
-to question 'I200', you answered 'Yes'

I201 Please let us know which ones _____

07 Quality Assurance

Please assess..	1 Strongly Agree	2 Agree	3 Uncertain	4 Disagree	5 Strongly Disagree
Q1 *The Open Source development approach is a guarantee for high quality products	○	○	○	○	○
Q2 *Well defined processes can improve the software quality	○	○	○	○	○
Q3 *User debugging delivers many defects reports in good quality	○	○	○	○	○
Q4 *Web Portals are best platform for OSS development projects	○	○	○	○	○
Q5 *Source Code Control Tools can improve software quality	○	○	○	○	○
Q6 *Code viewers can support the review process	○	○	○	○	○
Q7 *Automatic builds ensure high software quality	○	○	○	○	○
Q8 *Mailing Lists ease the communication	○	○	○	○	○
Q9 *Defect handling efficiency is increased with bug tracking tools	○	○	○	○	○
Q10 *Testing suites are mandatory for efficient testing	○	○	○	○	○
Q11 *Instant messaging support the direct project communication	○	○	○	○	○
Q12 *Frequent release cycle have a positive influence on the product quality	○	○	○	○	○
Q13 *Reusability of code leads to increased product quality	○	○	○	○	○
Q14 *Peer-reviews are as effective as normal reviews or inspections	○	○	○	○	○
Q15 *User testing influences the quality significantly	○	○	○	○	○

Q16 What do you think could highly improve the software quality in OSS development?

_____

Q17 *Do you perform or plan Quality Assurance activities within the project?
Please choose.. ○ Yes ○ No

[Only answer this if the following conditions are met:]
-to question 'Q17', you answered 'Yes'

Q18 Please specify briefly your QA processes or practices

_____

Q19 *Does the Quality Assurance team check that the product adheres to standards and requirements?
Please choose.. ○ Yes ○ No

Q20 *Did the Quality Assurance team trigger actions, resulting from reviews that affect the team or processes?
Please choose.. ○ Yes ○ No

[Only answer this if the following conditions are met:]
-to question 'Q20', you answered 'Yes'

Q21 Please specify briefly the actions performed

_____

*Please note that questions marked with a * are mandatory questions*

A.3 Survey Execution

Overview about the survey execution and the survey response history:

Table A.2. Survey Statistics

Wave	Records	Target Group	Execution	Reminder
1	3176	First listed person of each project	13.06.2007	18.06. 1st 25.06. 2nd
2	396	Only midsize projects with 10 or more developer	13.06.2007	18.06. 1st 25.06. 2nd
3	112	Only large projects with 20 and more developer	13.06.2007	18.06. 1st 25.06. 2nd
4	41	Only very large developer with 30 and more developer	13.06.2007	18.06. 1st 25.06. 2nd
5	2275	Second listed person of each project	25.06.2007	02.07. 1st
Total	**6000**			

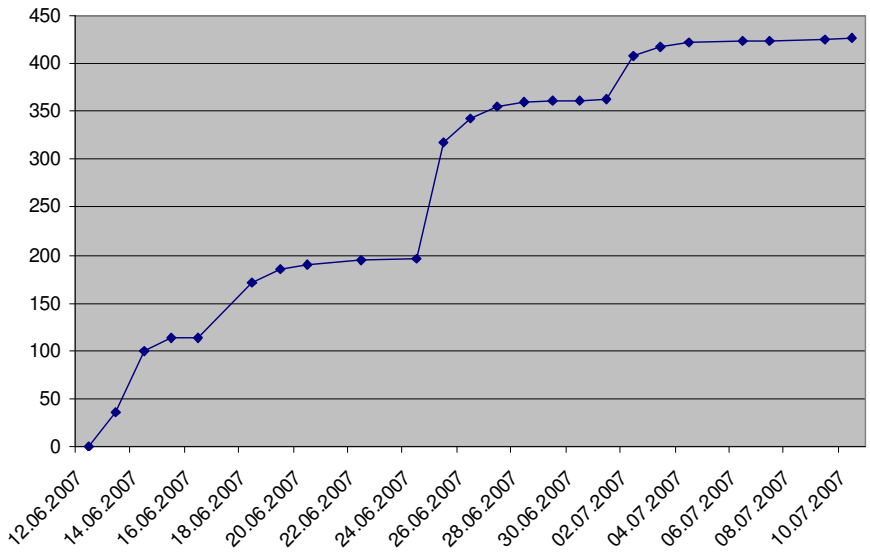

Figure A.1. Survey Response Histories

A.4 Survey Results

The following graphics show the results of the analysed tool in relation to project size.

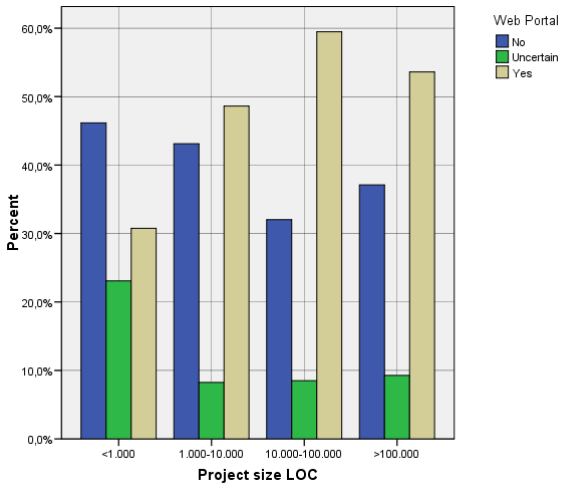

Figure A.2. Usage of Web Portals in relation to Project Size

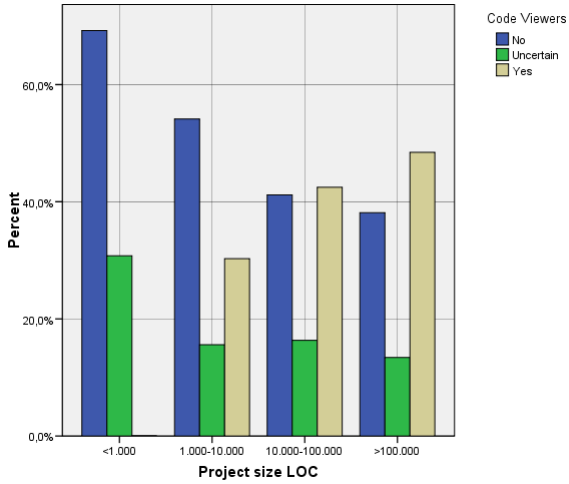

Figure A.3. Usage of Code Viewers in relation to Project

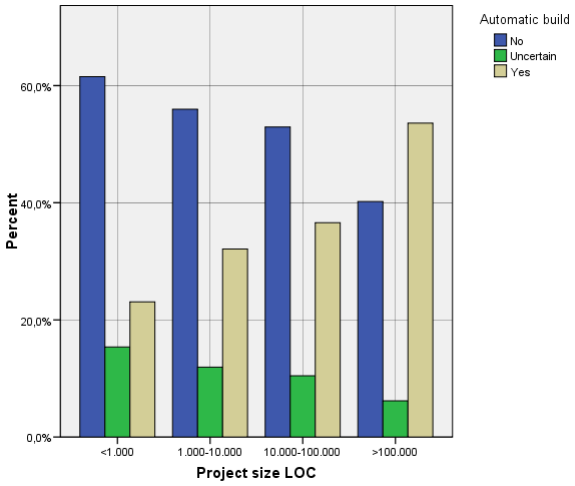

Figure A.4. Usage of Automatic Build Tools in relation to Project Size

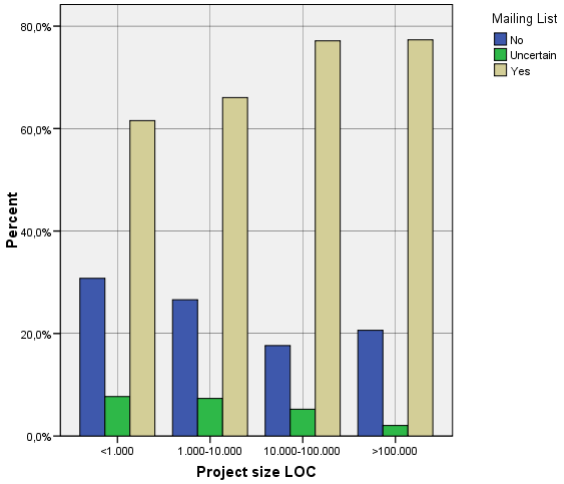

Figure A.5. Usage of Mailing Lists in relation to Project Size

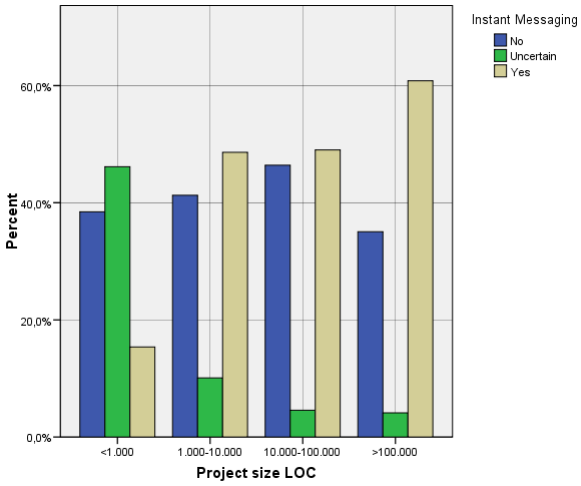

Figure A.6. Usage of Instant Messaging Tools in relation to Project Size

Table A.3. Overview about tool usage

Web Portal		Source Code Control		Code Viewer		Automatic Build Tool	
Tool	**%**	**Tool**	**%**	**Tool**	**%**	**Tool**	**%**
SourceForge	47,73 %	Subversion	56,77 %	ViewVC	17,39 %	Ant	21,32 %
Wiki	15,45 %	CVS	34,90 %	Subversion	13,66 %	Custom	11,17 %
Custom	10,45 %	Other	8,33 %	Trac	8,70 %	Make	9,14 %
Trac	5,00 %			Eclipse	8,07 %	Maven	7,61 %
Google	2,73 %			SourceForge	7,45 %	CruiseControl	6,60 %
Plone	2,27 %			CVSView	5,59 %	Eclipse	3,05 %
Other	16,36 %			Fish Eye	3,73 %	Autoconf	2,54 %
				Other	35,40 %	Automake	2,54 %
						Bamboo	2,54 %
						Other	33,50 %

Mailing Lists		Bug Tracking Tools		Test Support Tool		Instant Messaging Tools	
Tool	**%**	**Tool**	**%**	**Tool**	**%**	**Tool**	**%**
SourceForge Mailman	76,37 %	SourceForge BugTracker	46,18 %	JUnit	28,83 %	IRC	21,36 %
Custom	3,08 %	Bugzilla	16,88 %	Custom	20,25 %	MSN Messanger	18,31 %
Google Groups	2,40 %	Trac	8,60 %	CPPUnit	3,07 %	Skype	11,86 %
majodomo	2,40 %	Jira	7,32 %	DUnit	3,07 %	GoogleTalk	9,49 %
YahooGroups	2,40 %	Mantis	4,78 %	Harness	2,45 %	Gaim	8,81 %
Other	13,36 %	Custom	2,55 %	homemade	1,84 %	Jabber	8,14 %
		Other	13,69 %	Other	40,49 %	ICQ	6,78 %
						Yahoo	5,76 %
						AOL IIM	3,73 %
						Other	5,76 %

B. Case Study Approach

B.1 Case Study Interview Questionnaire

Interview Questions

The interview is part of PhD research program at University of Wolverhampton. The research focuses on the investigation of quality assurance processes in the open source development model. The aim of this interview is to gain a better understanding what kind of processes or "best practices" are applied in your project. The results are kept confidential. Any further analysis is done anonymous.

The interview consists of general and process related questions with focus on QA activities. Please answer the questions brief and state only headwords.

General Information

- What is the name of project you contribute the most to?
- What is the topic of the project?
- How many lines of code (LOC) does the project have?
- How many active developers contribute to the project?
- How many users participant in the project?
- What is the actual project status? (productive?)
- When did the project release the actual version?
- What is your role in the project?
- What are your main activities?

Processes

- How do you manage your requirements?
- Do you perform requirements reviews?
- What kind of process documentation exists?
- What kind of product documentation is available?
- How is your project organisation set up?
- How do you coordinate the project?
- How is team communication established?
- What kind of knowledge capturing do you perform?
- How do you ensure team education?
- What kind of infrastructure guidelines does exist?
- How do you check the design / specifications?
- How do you check that standards are kept during coding?
- How do you check code quality before commit?
- What types of reviews / inspections do you perform?
- Do you perform peer reviews?
- How is defect management handled?
- What kind of testing do you perform?
- How is release management handled?
- What kind of quality check do you perform before release?
- What general quality targets do you have?
- How is Software Quality Assurance performed?

- How do you improve your processes?
- Finally, please estimate the project success on a scale from 1 - 10, where 10 means a superior success and 1 means a flop?

B.2 Case Study Process Determination

Domain	Process Group / Process	Purpose	Outcome	Practices	*Examples*
Management	**Requirement Management**	*The process group "Requirements Management" focuses on capturing, management, control and review of project requirements. The continuous requirements evaluation under the OSSD demands an extended focus on requirement management to ensure a permanent reflection of the actual needs. The correct capturing of the requirement is basis for the definition of product quality characteristics and the acceptance of the product in the market.*			
	Requirement Management	Management of product requirements	- Capturing and documentation of product requirements ensured; - Process for requirements management established	- Continuous requirements capturing; - Perform requirements analysis; - Establish documentation and tracking of requirements; - Establish change management processes; - Introduce a requirement management tool; - Record changes	*Core team continuously captures user requirements and documents any feature requests in a public list.*
	Requirements Review	Review product requirement	- Product requirements are documented and up to date	- Conduct requirements review; - Incorporate changes into project planning; - Review changes	*Core team verifies requirements changes and critically discusses their implementation regarding conflicts or implications*
Management	**Documentation Management**	*This process group focuses on the product and process documentation. Process documentation describes mainly internal project processes, such as the development approach and applied standards, while product documentation comprises architecture or design specifications as well as user documentation or training materials.*			
	Process Documentation	Ensure documentation of processes, standards and templates; Establish collaborative accessibility to documentation	- Collaboration and software development processes are described; - Development style and coding guidelines are available	- Describe development processes, rules and guidelines; - Prepare development style and coding guidelines; - Develop templates to standardise development activities; - Continuously review process documentation	*Development style and coding guidelines are defined and online available*
	Product Documentation	Ensure adequate product documentation; Establish collaborative accessibility to documentation to ease collaboration and knowledge transfer	- Description of software engineering relevant documentation regarding requirements, specification, design, architecture, configuration; - Description of user relevant documentation regarding functionality, handling and maintainability of the product	- Plan and schedule required documentation; - Prepare documentation according to plan; - Perform reviews	*API documentation is maintained by the developer; user guidelines are setup; sample training documentation is online available*
Organisation	**Coordination and Team Communication**	*The process group "Coordination and Team Communication" focuses on the intergroup exchange of information. The establishment of an effective communication is regarded as an essential element in a distributed development environment to spread information within the community but also to ensure the daily communication in the development process. The coordination perspective focuses on the task tracking and control of results. It presupposes the establishment of suitable project organisation with roles and hierarchies.*			
	Project Organisation	Ensure effective organisation structures	- Organisational structures are defined and established	- Define organisational project structure with an appropriate hierarchy; - Setup communication and collaboration guidelines; - Establish OSSD role model and define authorisations; - Tailor role model according to project needs; - Review and adjust model structure	*Definition of roles, such as user, developer, core developer, release manager, project management, with different access rights to the repository*
	Project Coordination	Coordination of project tasks is established and results are controlled	- Project tasks, issues, work packages are accessible for every team member; - Results are checked and tracked	- Establish and document coordination rules; - Define work packages / tasks for software engineering activities; - Ensure accessibility to all tasks for all participants; - Force collaborative tracking of accepted tasks; - Ensure open information and discussion about tasks; - Check and review results by participants and core team	*Setup and maintenance of a public issue tracker; core team continuously reviews changes*
	Team Communication	Ensure effective team communication and exchange of information within the community	- Rules, guidelines and infrastructure for communication are defined and established	- Define communication rules and guidelines; - Establish appropriate communication tools (mailing lists and instant messaging); - Establish proper communication cycle (recurring tasks)	*Implementation of mailing lists or use groups; Defined guidelines are available on the web portal*
Resources	**Knowledge Transfer**	*The Knowledge Transfer processes focus educational and knowledge capturing activities within a project. The ability to effectively transfer know-how is a key element to counteract the lack of knowledge due to fluctuation. An efficient knowledge transfer process supports to take new developers on the project and is a foundation for the team education. The establishment of knowledge transfer processes is the one cornerstone increasing the capability of an organisation towards a high professional knowledge about e.g. product, processes, methods or development approach within the team.*			
	Knowledge Capturing	Counteract lack of project know how due to fluctuation	- Knowledge management and documentation procedures	- Ensure up-to-dateness of product and process documentation; - Ensure documentation of source code; - Ensure documentation of each work package before commit; - Continuous review of documentation	*Code commit to repository without appropriate documentation is rejected; Core team checks status of documentation and addresses update tasks to the community*

	Team Education	Improvement of varying skill levels within team; Preventative measure to avoid defects and shortcomings	- Professional knowledge about processes, approach, methods, rules, guidelines, standards, etc. across the team ensured; - Application of a professional development approach; - Training activities planned and conducted	- Analyse/monitor development skill level; - Prepare training or knowledge transfer documentation; - Conduct team training or ensure education through knowledge transfer documents; - Continuous review of team skill level	*Core team prepares training documentation or tutorials etc. and distributes/announces information via mailing list;* *"Best practices" are online available*
Resources	**Infrastructure and Tools**	*The process group "Infrastructure and Tools" regards the underlying support of processes, guidelines, standards, etc. by a proper environment. The definition and setup of a common project infrastructure, consisting of development and collaborative tools is a widespread solution how projects establish standards and common processes that reflect their development approach. On the other hand, collaboration tools realise processes. Especially in the distributed development environment, instant messaging techniques offer new possibilities and advantages for collaboration.*			
	Infrastructure Management	Define tools for entire lifecycle, which support the collaborative distributed development approach and reflect or enable project processes	- Software engineering processes effectively supported; - Collaboration processes supported; - Development processes reflected and supported	- Establishment of a freely accessible web portal; - Implementation of a source code control tool; - Recommendation of code viewers; - Setup of an automatic build tool, if appropriate; - Introduce a bug tracking tool; - Introduce a test support tool; - Initialise collaboration tools (mailing lists, instant messaging); - Describe guidelines and document project tool standards; - Review the effectiveness of the tool support	*Web Portal on e.g. Sourceforge established; CVS, mailing list setup, Bugzilla for bug tracking implemented; Jira for issue tracking; Skype for instant messaging implemented, etc.*
Development	**Software Engineering**	*The process group "Software Engineering" comprises the software development activities concentrating on design and source code development. The "Requirements Management" is precondition for software engineering. The design process describes all activities transforming requirements into the functional and/or technical specification. The source code development process builds upon the design process and describes the product development, such as source code or documentation. The process group requires an appropriate team communication, infrastructure and triggers the "Review and Inspections" and "Verification and Validation" processes.*			
	Design Control	Establishment of functional and technical product specification, which can be verified against the requirements	- Functional and technical design is defined and can be used for coding and testing	- Establishment of design documents, which are refined into lower levels that can be coded; - Consideration of modularity or reusability aspects; - Assurance of (internal/external) product quality characteristics; - Define testing requirements; - Continuous evaluation of design documents according to altering requirements; - Perform updates, adjustment of design documents and ensure consistency	*Solid architecture of the application is defined; class concept is established; design documents are available in the project Wiki*
	Development Control (Coding)	Development of the software product according to design documents	- Development of source code and documentation completed	- Development of source code according to development approach, methods, standards; - Keeping of internal and external quality goals; - Ensure documentation of source code and product documentation; - Perform unit testing and trigger Verification & Validation processes; - Submit source code to repository	*Developer contribute source code according to the project standards;* *Pair development is used to reduce defects;* *Quality standards are kept and the source code is proper documented*
	Continuous Code Quality Control	Continuously control the code quality	- Development standards for coding or documentation are kept; - Quality standard of source code is achieved	- Establish control and code release process before submit; - Ensure versioning and write access to repository; - Perform code reviews (quality control) by professional developers; - Perform corrective actions if code does not adhere quality standards; - Evaluate risks, side effects and adjust test planning accordingly; - Trigger peer reviews or inspections; - Rework or reject inappropriate source code; - Report findings and trigger SQA processes	*Each contributed source code needs to pass a unit test and consists of an appropriate documentation; code reviews are performed by core developers; inappropriate code is reworked or rejected*
Development	**Review and Inspections**	*The process group "Review and Inspection" describes a core process of reviewing the product development, especially the Open Source code in order to identify defects. Simple code reviews as well as peer reviews are recommended to improve the code quality.*			
	Code Review/Inspection	Identify defects, design gaps through reviews or inspections	- Code quality improved and defects identified	- Conduct code reviews; - Performs walkthroughs or code readings; - Perform corrective actions to adhere standards; - Record identified defects and trigger defect handling	*Each submitted piece of source code is reviewed by core developers before it is finally submitted to repository*
	Peer Review	Identify and remove defects due to peer reviews	- Improved code quality	- Plan peer reviews and establish the process; - Conduct peer reviews and record data; - Perform corrective actions and trigger defect handling	*Developers are forced to perform peer reviews in order to improve the quality of their deliverables*
Development	**Verification and Validation**	*The "Verification and Validation" process group describes the management and organisation of testing and defect handling activities. An effective defect management as well as stepwise conducted software testing with an acceptance strategy is of major importance. Software testing ought to comprise unit, integration, acceptance as well as regression testing before product release. As testing usually is performed within the community, main emphasis to test coordination and test organisation needs to be given. Both testing and defect handling processes require special importance regarding manageability, efficiency and understandability to take advantages from user testing. A continuous review of process efficiency is recommended.*			
	Defect Management	Management and control of defects along the entire lifecycle	- Defects are identified, clustered, managed and controlled	- Establishment of central defect reporting/tracking regarding requirements, design, source code, documentation; - Ensure defect classification and prioritisation as well as risk identification; - Address bug fixing tasks to team members; - Control and review bug fixing results; - Trigger quality control before code commit	*A bug tracking tool, such as Bugzilla is implemented;* *Issues are classified and prioritised;* *Public collaboration on removal of defects*

		Unit, Integration and Regression Testing	Identification of defects and verification of the product against the requirements	- Conduct appropriate testing to ensure that the product meets the requirements; - Defects are identified and corrective actions are triggered	- Define test strategy; - Suggest a test plan and appropriate test data; - Reflect varying product configuration and system architecture; - Define and document appropriate test environment; - Ensure effective test management and coordination; - Allocate development resources for fast defect removal time; - Establish mandatory unit testing by each developer; - Perform code reviews and peer review processes; - Ensure the establishment of defect handling processes; - Enable efficient user testing; - Plan and execute integration testing; - Plan and execute regression testing to ensure conformity of new features with existing environment; - Report and manage defect handling; - Perform retesting in case of defects	*A test support tool, such as JUnit is implemented; test cases are specified; mandatory unit testing is performed by each developer; code reviews are conducted; defects are reported to central defect management tool*
Management	**Release Management**	*The process group "Release Management" comprises the management, control and build of product releases. The focus of the release management process is the definition of the strategy, scoping and scheduling to deliver high software quality in a new release. An inappropriate management, scheduling or to wide scoping might lead to deliver incomplete, unstable or not adequately verified products. Therefore, release management requires the completion of verification and validation processes. The build and release check process focus on a pre- and post release quality control. The early delivery of a pre-final release "candidate" allows collaborative testing of the actual version to verify its stability, until the final release is built.*				
		Release Management	Management of release scope and scheduling to ensure stable releases	- Adequate release management strategy which reflects the development progress, the product maturity, requirements changes, scope enhancements or defect corrections	- Define release strategy (time and/or feature based); - Reflect product maturity and perform release quality check; - Identify requirements changes and scope enhancements; - Incorporate relevant corrections; - Verify release stability	*The release manager defines a feature based strategy; When main parts are ready a new release date is announced (time based). The development is concentrated on the completion of the activities till release date*
		Build and Release Check	Verification of requirements and release control	- Establish quality control to ensure defined product quality	- Perform pre- and post release quality control; - Perform verification and validation processes; - Distribute a pre-final release for collaborative testing; - Ensure compliance with product quality goals; - Verify compliance of product documentation;	*Implementation of nightly build (build tool); Community is forced for intensive testing (prior to release); release candidate is issued to check stability.*
Management	**Quality Management**	*The Quality Management process focuses on the establishment and management of quality goals and triggers the product quality measurement activities. The definition of product quality goals and the tailoring of appropriate measurement criteria to project needs are considered as major parts of Software Quality Assurance framework.*				
		Quality Management	Product or processes fulfil the quality objectives	- Quality management procedures are defined; - Quality targets are assured	- Define quality targets and standards; - Establish plans to achieve the quality targets; - Review quality targets and revise them; - Measure product quality; - Perform corrective actions	*Certain code modularity or complexity targets defined; Code coverage analysis are applied; Certain performance targets of application defined*
Management	**Software Quality Assurance**	*Software Quality Assurance (SQA) processes are considered as mandatory practices a project needs to fulfil in order to assure the process and product quality within the entire development lifecycle.*				
		Software Quality Assurance	Assurance of product and process quality according to plan	- Quality Assurance strategy is defined; - SQA activities are planned and performed; - Adherence of products and activities to the applicable standards, procedures, and requirements are verified objectively; - SQA activities/approaches are communicated to the project team; - Issues are addressed to the management	- Define QA strategy or plan; - Defines standards and procedures; - Setup QA tasks and plan; execution of activities; - Perform reviews to verify the adherence to standards; - Identify deviations from plan; - Perform measurements; - Report to team and management	*Project management team or community defines standards and reviews project deliverables. Reviews are triggered to verify the product quality. Quality assurance activities are communicated through mailing lists.*
Organisation	**Continuous Process Improvement**	*The Continuous Process Improvement processes reflect the capability of a team to transform a permanent improvement approach into the team behaviour and the willingness to critically question processes. Such mindset may contribute positively to the defect prevention approach. The ability of an organisation to reflect critically their current practices are considered as mandatory processes in mature organisations.*				
		Process Change Management	Continuous improvement of the software development process; Improvement of product and process quality	- Continuous improvement process is community wide established; - Improvement of product and process quality	- Define improvements targets; - Empower the team to improve processes; - Establish an improvement process within the community; - Evaluate and implement identified improvements; - Perform team training and knowledge management	*Core team or project management team forces the implementation of suggested process changes/enhancements by team members*
		Defect Prevention	Identification of cause of defects prevention of recurring defects	- Preventative measurements to avoid defects; - Common causes identified and systematically eliminated	- Defect prevention activities are planned; - Establishment of defect prevention process within the community; - Identification of causal relations; - Adaptation of processes and standards; - Record and review defect prevention activities	*Intensive discussions on mailing lists between developers how to prevent known issues in order to improve current processes*

B.3 Case Study Project Success Measurement

Sub characteristic	Attributes	*Purpose*	Metrics	Formula	Normalised Result (1-5)	Source
System creation and maintenance						
Activity / Effort	*File releases, CVS check-ins, mailing list discussions, tracker discussions, surveys of time invested.*					(Crowston *et al.*, 2006)
	Number of source code changes	*Determines the activity level to commit changes to the repository within the project*	Number of changes (commits) made in the source code until now.	= # of check-ins / # project months	5 - >50 4 - 21-50 3 - 11-20 2 - 5-10 1 - <5	(QualOSS D1.3 2007)
	Number of developer mailings	*Determines the activity level in public mailing lists*	Developers mailings / total number of messages	= # of developer mailings * 100 / total mailings	5 - >75% 4 - 61-75% 3 - 46-60% 2 - 25-45 1 - <25%	(QualOSS D1.3 2007)
	Issue Tracker activity	*Determines the activity level to record issues within the project*	Number of issues tracked or changed per month	= # of new issues plus # of changes / # project months	5 - >50 4 - 21-50 3 - 11-20 2 - 5-10 1 - <5	
Developer Attraction	*Attraction and retention of developers (developer satisfaction); size, growth and tenure of development team through examination of registration, CVS logs. posts to dev. mailing lists and trackers. Skill coverage of development team. Surveys of satisfaction and enjoyment.*					(Crowston *et al.*, 2006)
	Project size	*Determines the project complexity according LOC*	Number of Lines of Code	= total # of LOCs	1 minor <1.000 2 small 1.000-10.000 3 medium 10.001-100.000 4 large 100.001-200.000 5 very large > 200.000	Survey
	Developer team size	*Determination of actual development team size indicates the development potential of the project*	Number of active developers who have made changes in the project	= total # of active developers	1 - 1 2 - 2-5 3 - 6-10 4 - 11-20 5 - >20	Survey
	Developer community growth	*Determination of the growth of the developer community indicates the project attraction*	Number of new developers / total number of developers	= # number of new developers in the last 12 months * 100 / total # developers	5 - >40% 4 - 21-40% 3 - 6-20% 2 - 2-5% 1 - <2%	(QualOSS D1.3 2007)
	Developer fluctuation	*The number of inactive developers indicates the fluctuation within the project*	Number of inactive developers within the last year / total number of developers	= # number of inactive developers in last 12 months * 100 / total # of developers	5 - <5% 4 - 5-10% 3 - 11-20% 2 - 21-40% 1 - >40%	(QualOSS D1.3 2007)
	Continuity	*The number of commits and the quantity of changes in general as well as targeted to specific releases can be an indirect approximation of the continuity of effort put in the product*	Number of committed LOCs for all releases	= # of commits / # total number of releases	5 - >1000 4 - 501-1000 3 - 101-500 2 - 50-100 1 - <50	(QualOSS D1.3 2007)
Advancement of Project Status	*Release numbers or alpha, beta, mature self assessment, request for enhancements implemented,*					(Crowston *et al.*, 2006)
	Project maturity	*Determination of the maturity level of the project*	Release status	= actual release status	1 - Inactive/Planning 2 - Pre-Alpha 3 - Alpha 4 - Beta 5 - Productive	Survey
	Maturity capability	*Age of the first stable distribution release. It shows the capability to advance the product*	Required project months to reach the first mature status	= # of months required to reach beta status	5 - <12 3 - 12-18 1 - >18	
	Vitality	*Vitality describes the ratio of number of last releases multiplied by the duration compared to the total number of releases. It shows how vital the project was in the last year*	- R is the number of releases in a certain period (t) - a is the age of project (in days) - L stands for the number of releases in t	= (# of releases within the last 12 months * # project months (age)) / total # of releases	5 - >50 4 - 21-50 3 - 11-20 2 - 5-10 1 - <5	(Capiluppi *et al.*, 2003)
	Market availability	*Market availability is an indirect success factor*	Number of years in the market	= # of years in the market	1 - <0.5 year 2 - 0.5-1 year 3 - 1-2 years 4 - 2-4 years 5 - >4 years	Survey
Task Completion	*Time to fix bugs, implementing requests, meeting requirements (e.g. J2EE specification). time between releases.*					(Crowston *et al.*, 2006)
	Release frequency	*Determines how frequent releases are conducted*	Ratio of average time spent between releases	= # project months / # releases	5 - 6 3 - 4 - <6 or 6> - 12 1 - <4 or >12	(Ari 2007), (BRR 2006)
	Release activity	*Determines how many releases are conducted in the last year*	Number of minor releases in the last 12 months	= # releases in the last 12 month	5 - 2 3 - 1-3 1 - 0 or >3	(Ari 2007), (BRR 2006)

	Time to fix bugs	*Determines how quick defects are fixed. Low average bug fixing times indicate an easier correction of the defects and an easier maintenance of the actual release.*	Ratio of # of fixed bugs (in the last 6 months) / # open bugs (in the last 6 months)	= # of fixed bugs (in the last 6 months) * 100 / # open bugs (in the last 6 months)	5 - >75% 4 - 61-75% 3 - 46-60% 2 - 25-45% 1 - <25%	(BRR 2006)
	Short feedback loops	*Determines the duration for feedback between core team and community. It shows a direct reaction to fix defects.*	average # of days a reply to bug fixes or questions is given	= average # of days till response is given	5 - less 2 days 4 - less a week 3 - 1-2 weeks 2 - 2-3 weeks 1 - longer than 3 weeks	(Schmidt and Porter 2001)
Programmer Productivity	*Lines of code per programmer, surveys of programmer effort*					(Crowston *et al.*, 2006)
	LOC per developer	*Developer contribution in lines of code to the project*	Average LOC per developer	= total # of LOC / total # of developers	5 - >500 4 - 201-500 3 - 101-200 2 - 50-100 1 - <50	
	Developer community heterogeneity	*Determines the number of shared files between developers in the project. It shows if developers work on their own files or if there is a strong sharing, which increases the project complexity*	Number of people working on the same group of files	= total # of shared files / total # of active developers	5 - <1 3 - 1 1 - >1	(QualOSS D1.3 2007)
	Effective leverage of user community	*Determines the degree to which the project leverages of its developer community*	Number of changes submitted / total number of developers	= # of submitted changes / total # of developers	5 - >50 4 - 21-50 3 - 11-20 2 - 5-10 1 - <5	(Schmidt and Porter 2001)
Process Stability	*Development of stable processes and their adaption; Documentation and discussion of processes, rendering of processes into collaborative tools, naming of processes, adoption by other projects/endeavours; definition of development methodology*					(Crowston *et al.*, 2006))
	Process documentation	*Describes the degree to which processes are documented. It measures the manageability of project. It indirectly impacts taking new developers on the project*	Assessment of process documentation	Nominal rating	5 - detailed definition of processes, responsibilities and guidelines exist; 4 - some major project guidelines are available; 3 - only minor project descriptions exist; 2 - we manage the work mostly without process descriptions; 1 - processes are commonly known, no documentation exists	Survey
	Developer documentation	*Describes the degree to which clear development guidelines exists. It affects the on-boarding and development quality.*	Assessment of developer documentation	Nominal rating	5 - coding styles and development guidelines widely available; 3 - minor guidelines exist; 1 - no guidelines available	Survey
	Process automation	*Determines the degree that a tool support is necessary for an effective collaboration in the project*	Tools used within the project	Nominal rating	1 - no tools used 2 - basic tools such as CVS or mailing lists applied 3 - as 2 plus bug tracking tools applied 4 - as 3 plus test support tools used 5 - as 4 plus advanced tools applied	
Testing Effectiveness	*User testing effectiveness*					(Crowston *et al.*, 2006)
	LOC tested per user	*Determines the average amount of source code tested by users*	Ratio of tested source code to users	= # of LOC tested by users / total # of users	5 - >500 4 - 201-500 3 - 101-200 2 - 50-100 1 - <50	
	Average defects found per user	*Determines the average defects found by users*	Ratio of defects found by users	= # number of defects reported by users / total # of users	5 - >50 4 - 21-50 3 - 11-20 2 - 5-10 1 - <5	
	Efficiency of user testing	*Determination of user testing effectivity (reporting of critical defects by users)*	Ratio of critical defects determined by users in relation to all critical defects found	= # number of critical defects reported by users *100 / # of all critical defects	5 - >75% 4 - 61-75% 3 - 46-60% 2 - 25-45% 1 - <25%	Survey
Project Openness	*Open access to source code; open communication; easiness for new developers to join the project*					(Shaikh and Cerone 2007)
	Source code accessibility	*Determines the degree of accessibility and mastery of the source code*	Assessment of access options	Nominal rating	1 - no direct expertise of the code 3 - code accessible but mastery of code limited 5 - several individuals mastering code - widely accessible - including documentation	(QSOS 1.6 2007)
	Developer integration ability	*Measures the barriers for new developers joining to the project*	Assesses the degree of developer integration	Nominal rating	1 - strongly restricted joining (need to apply) 3 - restricted joining (ask for access) 5 - no restrictions at all - freely joining	(Kienzle 2001 in Rothfuss 2002)

	Communication openness	*Measures the degree of openness in project communication, .e.g. about issues, improvements, suggestions, etc. in mailing lists or working groups*	Assesses the degree of openness in the project communication	Nominal rating	1 - public communication exists, but non public communication strongly used 3 - main communication open, but private non public information exists 5 - all communication and information widely open	(Kienzle 2001 in Rothfuss 2002)
System quality						
Code Quality	*Code analysis metrics from software engineering, such as modularity, correctness, coupling, complexity*					(Crowston *et al.*, 2006)
	Modularity	*Modularity describes the split of the product into small parts and affects code reusability*	Average size of module = (code size/number of modules)	= # of LOC / # of modules	1 - >1000 2 - 501-1000 3 - 201-500 4 - 100-200 5 - <100	(Capiluppi *et al.*, 2003)
	Correctness	*Product correctness describes the ratio of correct results to the total results*	Absolute number of correctly implemented functions in relation to the total number of functions described in the requirements specification	= # correct implemented functions * 100 / # total required functions	5 - >75% 4 - 61-75% 3 - 46-60% 2 - 25-45% 1 - <25%	(ISO9126 2001)
	Coupling	*Determines the degree of coupling. Coupling can be "low" (also "loose" and "weak") or "high" (also "tight" and "strong"). Low coupling refers to a relationship in which one module interacts with another module through a stable interface and does not need to be concerned with the other module*	**For data and control flow coupling** di = number of input data parameters ci = number of input control parameters do = number of output data parameters co = number of output control parameters **For global coupling** gd = number of global variables used as	= (1 - 1/(di + 2·ci + do + 2·co + gd + 2·gc + w + r)) ***100**	5 - <75% 3 - 75-90% 1 - >90%	(Pressman 1992)
	Complexity	*Complexity is rated by the degree of LOC per method. If the number of lines of code per method is high - the methods are quite long and hence less easy to grasp. If the ratio is low, the product is less complex to understand, and hence easy to maintain.*	Average number of lines of code per method	= # LOC / # methods	1 - >500 2 - 201-500 3 - 101-200 4 - 50-100 5 - <50	(QualOSS D1.3 2007)
Manageability	*Time to productivity of new developers, amount of abandoned code*					(Crowston *et al.*, 2006)
	On-boarding time	*Indicates manageability; Efficient knowledge transfer*	Average number of weeks required to work productive on the project	Ordinal rating	5 - less a week 4 - 1 week 3 - <1 - 2 weeks 2 - <2 - 3 weeks 1 - 4 weeks and more	Survey
	Amount of abandoned code	*Indicates complexity level and maintainability of the projects; Indicates manageability*	Proportion of abandoned code to total LOC	= # abandoned LOC * 100 / # total LOC	5 - <5% 4 - 5-10% 3 - 11-20% 2 - 21-40% 1 - >40%	Survey
Documentation Quality	*Use of documentation, user studies and surveys.*					(Crowston *et al.*, 2006)
	Source code comments	*Determines the number of comments in the source code. If there is a high number of lines of comments, the coverage is higher.*	Ratio between the total number of lines of comments and the total number of lines of code in the package distribution list.	= total # lines of comments * 100 / total # LOC	5 - >40% 4 - 21-40% 3 - 11-20% 2 - 5-10% 1 - <5%	(QualOSS D1.3 2007)
	Outdated code documentation	*Determines the degree of outdated source code documentation. It measures the amount of outdated comments in code that should be removed. The average ratio comment to code is then lower as originally calculated.*	Ratio between the number of lines of comments commenting outdated code in relation to the total number of lines of comments	= # of lines of comments commenting outdated code * 100 / total # of lines of comments	5 - <5% 4 - 5-10% 3 - 11-20% 2 - 21-40% 1 - >40%	(QualOSS D1.3 2007)
	Inadequate code documentation	*Evaluates the quality of the documented source code. If there is a lot of comment lines that are not in line with the environing code, the comments are not commenting actual code.*	Ratio between the number of lines of comments not related to the environing code and the total number of lines of comments	= # of inadequate lines of comments * 100 / total # of lines of comments	5 - <5% 4 - 5-10% 3 - 11-20% 2 - 21-40% 1 - >40%	(QualOSS D1.3 2007)
	User documentation up-to-dateness	*Determines the actuality of the documentation to the actual release. If the user documentation is far older than the product release date, the chance is high that this user documentation is no more in line with the real content of the product release.*	Difference between the date of the user documentation and the date of the project release.	= week of documentation release - week of project released	5 - 1 4 - 1-2 3 - 2-4 2 - 4-8 1 - >8	(QualOSS D1.3 2007)

System use						
User Satisfaction	*User ratings, opinions on mailing lists, user surveys*					(Crowston *et al.*, 2006)
	User satisfaction checking	*Measures the user satisfaction based on an estimated rating*	Estimated average user satisfaction	Rating of project user satisfaction based on e.g. rankings or mailings	5 - excellent 4 - good 3 - acceptable 2 - poor 1 - unacceptable	
Number of Users	*Surveys (e.g. popularity contest), downloads, inclusion in distributions, package dependencies, reuse of code*					(Crowston *et al.*, 2006)
	Community size	*Determination of project size; Number of user shows if enough potential is available for code reviews*	Number of users	= total # of user	1 - <10 2 - 10-50 3 - 51-100 4 - 101-200 5 - >200	Survey
	Community size growth	*Determination of community members growth*	Number of new users in the last 12 months / total number of users	= # number of new users in the last 12 months * 100 / total # users	5 - >40% 4 - 21-40% 3 - 6-20% 2 - 2-5% 1 - <2%	(Kienzle 2001 in Rothfuss 2002)
	Downloads	*Determination of the popularity, expressed in downloads of the product*	Number of downloads	= total # downloads	1 - <10 2 - 10-50 3 - 51-500 4 - 501-1000 5 - >1000	(QualOSS D1.3 2007)
	Mailing interests	*Determines the amount of users interests in the projects in terms of entries in the mailing list*	Number of users writing in the mailing lists of the project	= total # of user mails / total # users	1 - <10 2 - 10-25 3 - 26-50 4 - 51-100 5 - >100	(QualOSS D1.3 2007)
Interest	*Site page views, porting of code to other platforms, development of competing products or spin-offs*					(Crowston *et al.*, 2006)
	Popularity	*Determines the popularity with regards to visits and subscribers*	- U stands for the count of visits to the project home page - R is the number of visits to the project on the web portal - S is the subscribers number	$P = \sqrt[3]{\frac{U * R * (S+1)}{3}}$	1 - <100 2 - 100-500 3 - 501-1000 4 - 1001-2000 5 - >2000	(Capiluppi *et al.*, 2003)
Support Effectiveness	*Number of questions effectively answered, time required to assist newbies*					(Crowston *et al.*, 2006)
	Questions answered	*Shows effectivity of answered questions*	Evaluation of number of answers in mailing lists	= # open questions * 100 / total # of questions	5 - <10% 4 - 10-20% 3 - 21-40% 2 - 41-60% 1 - >60%	(QualOSS D1.3 2007)
	Implemented requests	*Determines the number of implemented requests within the last 6 months*	Ratio of # implemented requests (in the last 6 months) / # of suggested (new) requests (in the last 6 months)	= # implemented requests * 100 / # suggested requests	5 - >75% 4 - 61-75% 3 - 46-60% 2 - 25-45% 1 - <25%	(QualOSS D1.3 2007)
	Support effectiveness	*Measures activity in mailing lists, especially of developers who participate in these mailing lists in order to provide a lot of information to normal users*	Number of developers replying in mailing lists in relation to total number of developers	= # of developer replying mail * 100 / total # of developers	5 - >75% 4 - 61-75% 3 - 46-60% 2 - 25-45% 1 - <25%	(QualOSS D1.3 2007)

B.4 Case Study Process Capability Determination Results

Table B.1. Case Study Process Capability Determination - Summary

Area	Domain	CASE 1	CASE 2	CASE 3	CASE 4	CASE 5	CASE 6	CASE 7
Management	**Requirements Management**	26	26	20	19	23	19	9
	Documentation Management	23	23	29	28	19	20	6
	Quality Management	4	11	9	0	6	8	2
	Software Quality Assurance	7	0	12	13	8	8	4
	Release Management	16	26	26	30	24	31	28
Organisation	**Coordination and Communication**	37	39	35	41	39	37	31
	Continuous Process Improvement	2	21	11	6	23	12	0
Resources	**Knowledge Transfer**	9	18	15	21	19	14	9
	Infrastructure and Tools	13	13	15	15	14	16	14
Development	**Software Engineering**	42	42	36	21	32	32	26
	Review and Inspections	18	29	10	24	25	17	25
	Verification and Validation	29	24	30	28	23	19	8
		226	**272**	**248**	**246**	**255**	**233**	**162**

The table shows the detailed results (including the weighting factor)

Table B.2. Case Study Process Capability Determination - Details

Area	Domain	CASE 1		CASE 2		CASE 3		CASE 4		CASE 5		CASE 6		CASE 7	
		R	*W*	R	*W*	R	*W*	R	*W*	R	*W*	R	*W*	R	*W*
Management	Requirement Management	14	*4*	13	*5*	11	*3*	11	*3*	13	*4*	9	*4*	6	*4*
	Requirements Review	12	*5*	13	*5*	9	*3*	8	*2*	10	*4*	10	*3*	3	*3*
	Process Documentation	12	*4*	13	*5*	16	*5*	15	*5*	8	*4*	11	*3*	2	*2*
	Product Documentation	11	*5*	10	*5*	13	*4*	13	*4*	11	*5*	9	*5*	4	*4*
	Quality Management	4	*2*	11	*4*	9	*2*	0	*2*	6	*3*	8	*3*	2	*1*
	Software Quality Assurance	7	*3*	0	*2*	12	*3*	13	*2*	8	*5*	8	*3*	4	*2*
	Release Management	11	*5*	13	*5*	15	*3*	15	*3*	11	*4*	16	*5*	14	*4*
	Build and Release Check	5	*3*	13	*5*	11	*3*	15	*3*	13	*5*	15	*5*	14	*4*
Organisation	Project Organisation	10	*3*	13	*4*	11	*3*	15	*3*	14	*4*	13	*4*	8	*3*
	Project Coordination	14	*4*	13	*4*	11	*4*	11	*4*	15	*5*	14	*4*	10	*4*
	Team Communication	13	*4*	13	*5*	13	*4*	15	*3*	10	*3*	10	*5*	13	*5*
	Process Change Management	2	*2*	13	*5*	8	*2*	0	*1*	13	*5*	3	*2*	0	*2*
	Defect Prevention	0	*1*	8	*5*	3	*2*	6	*2*	10	*5*	9	*4*	0	*1*
Resources	Knowledge Capturing	4	*4*	9	*5*	13	*3*	10	*3*	8	*5*	6	*5*	7	*3*
	Team Education	5	*3*	9	*5*	2	*2*	11	*2*	11	*5*	8	*2*	2	*2*
	Infrastructure Management	13	*4*	13	*5*	15	*5*	15	*4*	14	*5*	16	*5*	14	*4*
Development	Design Control	12	*3*	13	*5*	13	*3*	0	*2*	8	*5*	5	*3*	3	*2*
	Development Control	14	*4*	13	*5*	11	*3*	10	*2*	13	*5*	16	*5*	10	*4*
	Continuously Code Quality Control	16	*5*	16	*5*	12	*4*	11	*3*	11	*5*	11	*3*	13	*4*
	Code Reviews/Inspections	11	*3*	16	*5*	6	*3*	11	*3*	11	*5*	11	*3*	12	*5*
	Peer Review	7	*2*	13	*5*	4	*2*	13	*3*	14	*5*	6	*2*	13	*5*
	Defect Management	14	*4*	13	*5*	16	*5*	15	*4*	15	*5*	16	*5*	5	*4*
	Unit, Integration and Regression Testing	15	*5*	11	*5*	14	*5*	13	*5*	8	*5*	3	*3*	3	*2*
		226	82	272	109	248	76	246	68	255	106	233	86	162	74

B.5 Case Study Project Success Determination Results

Table B.3. Case Study Project Success Determination

Area	Sub-characteristics	Attributes	Initial	CASE 1		CASE 2		CASE 3		CASE 4		CASE 5		CASE 6		CASE 7	
			W_i	R	*W*	R	*W*	R	*W*	R	*W*	R	*W*	R	*W*	R	*W*
System creation and maintenance	Activity / Effort	Number of source code changes	5	5	*1*	5	*5*	5	*2*	5	*2*	5	*3*	5	*1*	3	*2*
		Number of developer mailings	4	4	*2*	5	*2*	5	*2*	4	*1*	3	*3*	2	*1*	3	*1*
		Issue Tracker activity	4	5	*2*	1	*5*	5	*1*	5	*2*	5	*4*	5	*1*	2	*1*
	Developer Attraction	Project size	1	3	*2*	3	*1*	3	*2*	5	*2*	5	*4*	4	*2*	3	*2*
		Developer team size	1	3	*2*	3	*1*	3	*2*	5	*2*	4	*5*	2	*3*	2	*3*

		Developer community growth	**3**	3	*1*	3	*4*	2	*2*	2	*1*	3	*4*	3	*2*	1	*4*
		Developer fluctuation	**2**	5	*1*	2	*3*	4	*5*	4	*2*	5	*4*	2	*1*	1	*4*
		Continuity	**3**	5	*1*	3	*1*	3	*2*	5	*3*	5	*4*	5	*1*	1	*3*
	Advancement of Project Status	**Project maturity**	**5**	5	*5*	5	*5*	5	*5*	5	*3*	5	*5*	5	*5*	4	*2*
		Maturity capability	**4**	5	*4*	1	*1*	5	*4*	5	*1*	3	*4*	5	*5*	3	*1*
		Vitality	**3**	3	*3*	0	*1*	4	*4*	2	*1*	4	*5*	3	*3*	3	*1*
		Market availability	**4**	5	*5*	5	*4*	5	*5*	5	*3*	5	*5*	5	*5*	5	*2*
	Task Completion	**Release frequency**	**5**	3	*2*	3	*4*	3	*3*	1	*2*	1	*4*	3	*2*	1	*4*
		Release activity	**4**	3	*3*	3	*1*	1	*2*	5	*2*	1	*5*	5	*2*	1	*4*
		Time to fix bugs	**5**	5	*2*	5	*5*	5	*3*	5	*3*	4	*5*	5	*3*	2	*3*
		Short feedback loops	**5**	4	*4*	5	*5*	4	*4*	4	*3*	4	*5*	4	*4*	5	*3*
	Programmer Productivity	**LOC per developer**	**4**	5	*4*	3	*1*	5	*4*	5	*2*	4	*2*	5	*1*	5	*3*
		Developer community heterogeneity	**2**	1	*1*	0	*0*	3	*2*	1	*1*	1	*3*	1	*1*	1	*3*
		Effective leverage of user community	**5**	5	*1*	2	*3*	3	*1*	5	*1*	2	*5*	5	*1*	1	*3*
	Process Stability	**Process documentation**	**5**	4	*5*	5	*5*	5	*4*	5	*3*	4	*5*	4	*4*	3	*2*
		Developer documentation	**4**	5	*5*	5	*5*	5	*5*	5	*3*	3	*5*	4	*4*	3	*1*
		Process automation	**4**	5	*4*	2	*5*	4	*4*	5	*3*	5	*5*	5	*5*	3	*2*
	Testing Effectiveness	**LOC tested per user**	**5**	1	*2*	0	*1*	3	*2*	0	*0*	3	*4*	1	*2*	3	*3*
		Average defects found per user	**5**	2	*2*	1	*5*	3	*1*	0	*0*	3	*4*	1	*3*	1	*4*
		Efficiency of user testing	**4**	1	*2*	4	*4*	4	*1*	0	*0*	3	*5*	1	*5*	3	*4*
	Project Openness	**Source code accessibility**	**3**	3	*4*	4	*5*	3	*5*	3	*2*	5	*5*	3	*3*	5	*4*
		Developer integration ability	**3**	1	*3*	5	*5*	1	*3*	1	*2*	3	*4*	1	*3*	1	*3*
		Communication openness	**3**	3	*3*	5	*5*	5	*3*	3	*1*	3	*4*	3	*3*	5	*3*
System quality	**Code Quality**	**Modularity**	**4**	1	*1*	3	*5*	4	*3*	0	*0*	3	*5*	1	*2*	3	*2*
		Correctness	**3**	5	*2*	5	*5*	5	*4*	0	*0*	5	*5*	4	*2*	3	*4*
		Coupling	**2**	4	*1*	3	*2*	3	*2*	0	*0*	3	*4*	0	*1*	5	*3*
		Complexity	**4**	1	*1*	2	*5*	3	*2*	0	*0*	4	*5*	3	*2*	4	*3*
	Manageability	**On-boarding time**	**4**	4	*5*	2	*4*	3	*1*	2	*2*	3	*5*	4	*4*	3	*2*
		Amount of abandoned code	**5**	5	*2*	5	*5*	5	*4*	0	*0*	4	*5*	5	*2*	4	*3*
	Documentation Quality	**Source code comments**	**3**	2	*1*	2	*5*	4	*3*	0	*0*	3	*4*	2	*3*	4	*4*
		Outdated code documentation	**3**	5	*3*	5	*5*	5	*4*	0	*0*	5	*5*	5	*2*	4	*3*
		Inadequate code documentation	**2**	5	*4*	5	*5*	4	*5*	0	*0*	4	*5*	5	*1*	3	*3*
		User documentation up-to-dateness	**3**	1	*4*	1	*3*	3	*4*	0	*0*	4	*5*	1	*5*	1	*3*
System use	**User Satisfaction**	**User satisfaction checking**	**3**	4	*5*	5	*5*	4	*5*	0	*0*	4	*5*	4	*5*	4	*4*
	Number of Users	**Community size**	**4**	3	*3*	5	*5*	5	*5*	5	*2*	5	*5*	5	*4*	5	*3*
		Community size growth	**2**	2	*3*	3	*5*	3	*2*	0	*0*	3	*5*	3	*4*	3	*3*
		Downloads	**3**	5	*4*	5	*5*	5	*5*	5	*2*	5	*5*	5	*3*	5	*3*
		Mailing interests	**3**	5	*5*	1	*4*	5	*4*	0	*0*	5	*5*	1	*1*	2	*3*
	Interest	**Popularity**	**3**	5	*5*	3	*2*	5	*4*	0	*0*	5	*5*	5	*5*	5	*3*
	Support Effectiveness	**Questions answered**	**4**	5	*5*	5	*5*	5	*3*	1	*2*	3	*5*	5	*5*	3	*4*
		Implemented requests	**5**	4	*4*	5	*5*	5	*4*	0	*0*	4	*5*	2	*3*	2	*3*
		Support effectiveness	**4**	3	*4*	3	*5*	4	*3*	3	*2*	3	*5*	4	*3*	3	*4*
			169	**171**	**138**	**156**	**177**	**186**	**150**	**116**	**61**	**176**	**213**	**161**	**133**	**140**	**135**

B.6 Results of the Statistical Analysis

Third Analysis: Process Capability Score to selected weighted Project Success Score

Overview about the results of the statistical analysis of the process capability score percentage model coverage weighted (PCS_MCW) to project success score weighted selected (PSS_SEL):

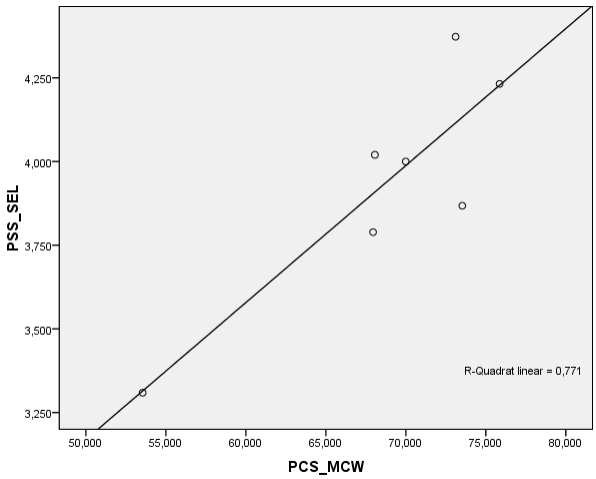

Figure B.1. Scatter plot of PCS_MCW to PSS_SEL

Table B.4. Model Summary – PCS_MCW to PSS_SEL

Model Summary(b)

Model	R	R-Square	Adjusted R-Square	Std. Error of the Estimate
1	,878(a)	,771	,725	,180110

a Predictors : (Constant), PCS_MCW
b Dependent Variable: PSS_SEL

Table B.5. ANOVA – PCS_MCW to PSS_SEL

ANOVA(b)

Model		Sum of Squares	df	Mean Square	F	Signifi-cance
1	Regression	,547	1	,547	16,852	,009(a)
	Residual	,162	5	,032		
	Total	,709	6			

a Predictors : (Constant), PCS_MCW
b Dependent Variable: PSS_SEL

Table B.6. Coefficients – PCS_MCW to PSS_SEL

Coefficients(a)

Model		Unstandardized Coefficients		Standardized Coefficients		
		B	Std. Error	Beta	T	Significance
1	(Constant)	1,122	,690		1,625	,165
	PCS_MCW	,041	,010	,878	4,105	,009

a Dependent Variable: PSS_SEL

C. Original Papers

The following original papers are included:

1. OTTE, T., MORETON, R. and KNOELL, H.D. (2008a) Applied Quality Assurance Methods under the Open Source Development Model. *in Proceedings 32nd Annual IEEE International Computer Software and Application Conference 2008, 28.7.-1.8.2008, Turku, Finland,* pp. 1247–1252.

2. OTTE, T., MORETON, R. and KNOELL, H.D. (2008b) Development of a Quality Assurance Framework for the Open Source Development Model. *in Proceedings Annual IEEE 3rd International Conference on Software Engineering Advances ICSEA 2008, October 26-31, 2008, Sliema, Malta,* pp. 123–131.

C.1Applied Quality Assurance Methods under the Open Source Development Model

Tobias Otte
University of Wolverhampton
School of Computing and Information Technology
Wolverhampton, UK
Tobias.Otte@wlv.ac.uk

Robert Moreton
University of Wolverhampton
School of Computing and Information Technology
Wolverhampton, UK
R.Moreton@wlv.ac.uk

Heinz D. Knoell
Leuphana University of Lueneburg
Lueneburg, Germany
Knoell@uni.leuphana.de

Abstract

Open Source Software (OSS) has reached widespread popularity within the last years not least because of renowned products such as Linux, the Apache Web Server or the Mozilla project. Under the Open Source Software Development (OSSD) model products are launched in rapid succession and with high quality, without following traditional quality practices of accepted software development models [1]. Furthermore, some OSSD projects challenge established Quality Assurance (QA) approaches, claiming to be successful through partially contrary techniques.
The aim of this research is to improve the understanding of QA practices under the OSSD model. A survey research method is used to gain empirical evidence about applied QA practices in midsize to large OSS projects. A further evaluation of successful projects results that they apply well-structured and organized development processes. The findings provide evidence for Raymond's lifecycle and show that OSS projects leverage their communities effectively.

1. Introduction

Open Source Software (OSS) is developed by freely participating programmers, who distribute source code in a collaborative, virtually and geographically distributed environment, communicating over the Internet [1], [2]. In contrast to proprietary software development, which is classified by plan, schedules, resources and deliverables, the OSS model starts with an idea, followed by a more prototypical approach with frequent release cycles. The OSS momentum is driven by the participant's motivation, the free availability of the source code, ongoing interactive tasks and a continuous feedback by the community. The Open Source Software Development (OSSD) model uses unconventional methods, such as the involvement of large development communities for coding. This contradicts Brooks' law [3], which faces an increased complexity due to exponential growth of communication and co-operation with an increasing project size. However, the OSSD model delivers successful products, such as Linux, Apache Tomcat or Mozilla, which appear to be high quality. The lasting success of the OSSD model delivering superior products makes it important to draw further attention on this phenomenon.
The research explores the following question: How is Quality Assurance (QA) under the OSSD model in mid- to large sized OSS projects conducted and what key practices do we learn from successful approaches?
In recent years, research about QA in OSS evolved, but only a few empirical studies exist that comprise the whole development lifecycle. This research paper explores applied QA practices and provides empirical evidence about software quality assurance methods in OSS projects. Furthermore, it analyses successful projects in order to find common patterns, which distinguish these projects and indicates key processes that contribute to their success. The findings may support actual OSS projects, identifying their weaknesses or supporting the improvement of their methods.

2. Quality Assurance under the OSSD model

Software Quality Assurance (SQA) is defined as a set of systematic activities providing evidence of the ability of the software process to produce a software product that is fit to use [4]. It is assumed that software quality is built into the product through the use of a process that has quality built in. Thus, SQA must be an aspect of all software development activities [5].
Fundamental empirical studies about QA activities under the OSSD model can be found in the literature by Zhao and Elbaum [6], [7], Halloran

and Scherlis [8], Koru and Tian [9] or Michlmayr [10]. The recent studies confirmed the uniqueness of the OSS model and provided evidence for Raymond's lifecycle. They show that user participation is extremely high, defect-handling processes follow mainly structured approaches, testing takes a significant portion of the software life cycle and there is a high usage of configuration and bug tracking tools. However, project documentation is often rare, design documents are lacking, source code may remain unmaintained when developers leave the project or testing face complexity issues, in case developers may have limited access to diverse platform configurations. Michlmayr [10] examined some OSSD processes in relation to project success and showed that projects that are more successful make more use of version control tools, systematic testing and effective communication through the deployment of mailing lists.

The studies provide useful information and constitute the basis for this research. An empirical view of QA practices within the development lifecycle in conjunction with project success measures is missing. Our survey was conducted combining quality characteristics with process success measures to identify key practices in OSSD projects.

3. Research Method

Our survey research is undertaken following an interpretative tradition using a social relativism paradigm [11]. The major objectives of the survey are quality criteria regarding development processes, defect handling and testing techniques, documentation as well as infrastructure and quality assurance criteria. The development of the questionnaire follows the recommended method by Grover [12]. The questionnaire was included project success measures as suggested by Crowston et al. [13]. Content validity and reliability is established through literature reviews, expert judges and a pre-test [14]. The survey questionnaire is available online at http://www.qafoss.org/data/survey.html.

The target group of the survey was individual developers or project managers contributing to mid- or large sized OSS projects. It is assumed that this unit of analysis could provide the most useful insights into applied practices and has a large practical experience. The target group is randomly chosen from pre-selected projects mainly hosted on Sourceforge.

A multimode approach is selected, which combines e-mail based communication with the participants and a web-based survey for data collection. The survey was executed in the period from 13th of June until 10th of July 2007. In total 427 participants responded to the survey, 11 records were incomplete or invalid, which provides a response rate of 8.2%. The main statistical analysis was done with SPSS 15.0, while some minor analyses were done with Excel.

4. Results

The multimode approach proved to be efficient. It follows the collaborative approach in a distributed development environment, using mailings and freely accessible online tools. Open-ended questions provided a detailed feedback about applied practices and further recommendations. In the following, the research results are discussed and the key aspects in conjunction to project success are shown.

4.1. General Information

The participating projects are grouped according to their size in lines of code (LOC) into the categories mini 3.1% (<1000), small 26.2% (1,000-10,000), medium 36.8% (10,000-100,000) and large 23.3% (>100,000). Around 10% of the respondents could not classify their project size for any reason. The project sizes are normally distributed. The main proportion of the projects belong to "Software Development" (20.7%) or "Internet" (17.8%), while other application types, such as Office, Database, ERP/Financials, Education, Networking, Entertainment or Communications are equally represented with a proportion of around 5% for each type. The respondents main role belongs to the group of developer (40.1%), project manager (38.5%) and engineering/design (13.2%), while 8.2% have others roles.

The majority of projects (55.8%) are developed by small groups of 2-5 core developers, 10.3% have only one developer, 17.8% of the sample have groups of 6-10 developer, 7.4% have 11-20 developers and 8.7% have more than twenty developers. These finding show evidence of much larger development groups as observed by Zhao and Elbaum [7].

Only 26.2% projects of the sample responded to have less than ten users, 33.4% have 10-50 users, 9.1% have 50-100 user and 31.3% argue to have more than 100 users. These figures base upon the participant's evaluation and represent only estimations. This study shows the existence of much smaller user groups compared to Zhao and Elbaum [7] who reported that 59% of the projects had more than 50 people, while this survey results 40.3%.

Evidence for a growth of project community with the project size could be observed. 86.5% of the mini projects stated to have less than 50 users. But there is a significant growth of the commu-

nity with an increasing project size, as 77.7% of projects with more than 20 developers stated to have a large community (<100 users).
Furthermore, projects have a major growth of their community size with the time in the market. Around 55% of the projects have less than 10 users in their first year, while 58.02% have more than 50 users in their fourth year.
The completion of project status from "beta" toward a productive status reaches 57.8% of the projects after the first year on the market.
The participant's level of knowledge in software development is extremely high, as more than 75% state to have above five years development experience. Massey [15] emphasizes the importance of an experienced team, as professional developers provide accurate feedback. The knowledge transfer to new participants lasts, on average 2-3 weeks. Only mini projects report an "On boarding" time of about one week, while larger projects need 4-5 weeks and more. The existence of knowledge transfer processes are more frequent with project growth. However, only 36% in large projects claim to have them.
In small development teams (2-5 developers), participants adopt multiple roles. Even in larger teams, projects participants have multiple roles, which confirms the findings of Jensen and Scacchi [16] that versatile and fluid roles are specific in the OSSD. Developers are mainly motivated by personal needs (36.5%), 27.2% by company needs, 22.1% by community needs and 14.2% by other reasons. However, in relation to project size company needs become more important with project growth. This shows a higher commercial interest in larger projects.
More than 54% of the participants work part-time. Nevertheless, large projects show a high proportion of full-time participants, which also corresponds to an increased company motivation in this area.
The survey indicates that developer fluctuation seems to be lower than expected, although OSS developers are freely participating or leaving. Mature projects, which are more than four years in the market report less than 1% of new participants with minor development experience (less one year). It can be assumed that participants attend projects over a long period, while participants experience matures with the project or simply projects attract only mature professionals.

4.2. Development

The average proportions of the development activities in the lifecycle, such as design (25%), coding (48.4%) and testing (26.6%) show an almost uniformly distributed picture independent to the project size. However, the proportion of coding effort increases to the disadvantage of testing with project growth.
Around 38.9% of the projects have code changes of about 10-20% between major releases as depicted in Figure 1. On average 24.3% of the code is reused by projects. Branches, which are up to 100% clones, might deform the result upwards. Roughly, 55% of the large projects considered modularity during design, while 58.3% of the mini projects reconsidered their code modularity during development and 8.3% had no modular development at all. That leads to the conclusion that mini projects start from scratch and may improve their modularity with product growth.

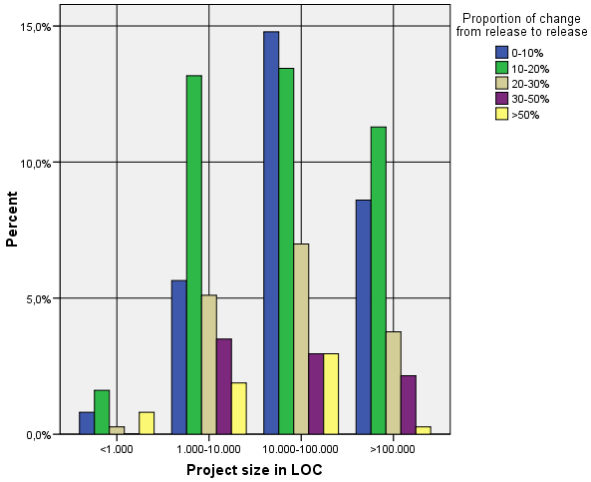

Figure 1. Code change between major releases

The estimated level of abandoned code averages about 12% per project. Projects with an increasing level of abandoned code reported more quality issues and side effects than others, as depicted in Figure 2. These results confirm Michlmayr [10] findings, who observed certain quality issues due to unsupported code.
Most of the projects (72.1%) follow a feature based release strategy, which is triggered by readiness of the code. However, there is a growing tendency of hybrid approaches as combination of time and feature based strategies. In general, the observed release frequency is lower than expected, as 5.3% of the projects once or twice every fortnight, 14.4% once a month, 29.6% once a quarter, 32.7% releases every 6 month and roughly 18% have even longer intervals. In contrast to Raymond's [17] often-cited statement, "release early and release often" the observed release frequency is much lower and contrasts with the study of Zhao and Elbaum [7] in which 43% of the projects release every month. On the other hand, too short release cycles could affect the software quality and frustrate end-users due to less stability [18].

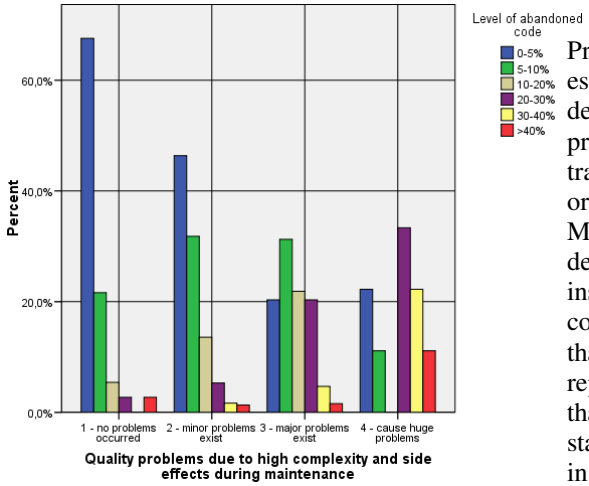

Figure 2. Code complexity to abandoned code

4.3. Testing

The average testing time compared to the whole development time was 38.6%. This reflects the findings of Zhang and Pham [19] who argued that projects spend 20-40% in testing. More than half of the projects (52.3%) follow a structured testing approach.

Around 57% of the projects reported a high user testing efficiency, as the users found major defects or hard bugs. The very positive team communication may contribute to this effect, as 72.9% of the participants reported a direct and efficient feedback between developer and user. Larger projects rate their communication as more effective than smaller ones. This contradicts Michlmayr [10] findings of communication problems in mid-size to large projects.

In comparison to code reviews by developers (47.2%), user testing (55.7%) seems to have the same or even a higher importance for projects. OSS projects frequently apply code readings (69.9%) or walkthroughs (61.4%). Peer reviews (50.5%) are not as frequently applied. However, Raymond [17] emphasized its importance, as "the high level of quality is partly due to the high degree of peer reviews and user involvement".

An important part of user involvement are user suggestions. 48.5% of all respondents mention that user suggestions bring the design forward, 49.3% feel that they are only sometimes useful, while 1.7% say they are not efficient or do not fit into the design (0.5%). It shows that OSS projects work as claimed by Raymond [17] and that they listen to their customers.

Around 46% of the projects perform corrective actions before a contributed code is committed to the repository. Large projects have strict quality checks, as 75% of them report rework or even reject inappropriate code.

4.4. Defect Handling

Projects either introduce defect-handling processes at project start (31.7%) or at start of the development activities (32.5%). The majority of projects (88.9%) track source code defects, while tracking of requirements (41.8%), design (43.0%) or documentation issues (48.5%) are less praised. More than 50% of the projects receive useful bug descriptions and only 13.8% complain about insufficient information. These findings show a contrary picture to Michlmayr [10] who argue that developers see an increase of useless bug reporting with less technical skilled users. More than 77% of the projects classify their defects for status tracking. This shows an improved situation in contrast to Villa [20], who observed absent information about priorities or severities for defect handling.

4.5. Documentation

The majority of projects (66.6%) use coding and development style guidelines, while 58% of the projects have common process documentation. However, process descriptions gain importance with project growth. For instance, 71% of large projects claim to have at least minor process documentation. More than 68% of the projects have product documentation or at least a draft. This study shows a far improved position compared to Michlmayr [10], who observe a lack of user and developer documentation.

4.6. Infrastructure

The observed tool usage in OSS projects is relatively high. For instance, 87.2% of the projects uses source code control tools, 76.2% bug-tracking tools and 73.5% mailing lists, 51.9% instant messaging, while only 36.5% apply test support tools. Projects that apply bug-tracking tools report an averagely higher defect reporting quality. For example, 50% of the projects that report not to get any useful information at all from their participants do not make use of a supporting tool. A further analysis of tool usage in relation to project size shows that large projects in general make more use of supporting tools, than smaller ones. For instance, 60% of the mini projects do not benefit from using bug-tracking tools.

4.7. Quality Assurance

Only one-third of the projects apply QA practices although they become more important with increased project size. An analysis of applied QA practices shows that several projects discuss quality issues within the core development team or their community. A number of projects con-

duct different kind of code reviews and implement structured test approaches, defect handling as well as test management. Quite a few projects established live sessions, smoke testing as well as automated nightly testing to improve software quality. Some projects restructure their organization and setup QA teams, who contribute full-time to the project.
Only 20% of the projects replied that their QA team performs actions, resulting from interviews or reviews. For instance, projects increase their communication activities, adopt their development approach and apply refactoring processes, rework flawed code, introduce additional code reviews, improve their defect handling and bug reporting processes, increase testing prior to a releases or even reconsider the release scope, such as the definition of exit criteria or stopping the release.
The respondents suggest the following quality improvements of the OSSD model: The enhancement of the community due to the attraction of knowledgeable users, professional full-time developers, high integration of users or developers and even paid contributors. Moreover, the project management needs to be experienced and has to provide leadership and guidance for their projects. The whole community must be conscious about quality to achieve significant improvements. However, large projects require an independent QA team. The QA team needs to perform checks and verify that processes or guidelines are kept, which can be supported by quality assessment tools. Projects should have detailed process documentation and development guidelines, comprising style or coding standards. A technical design phase prior to the development could reduce shortcomings in architecture or development approach. The release approach must be appropriate to the projects, regarding strategy, frequency and scoping. The respondents see room for improvement in testing processes, such as test efficiency, improvements of code reviews or the use of tools for automated testing. The tools should be OSS products themselves to lower barriers for communities. Furthermore, tools should focus on a higher integration, while there is another demand for easy handling and simplicity.

5. Project Success Examination

The measurement of project success is an elusive target and depends not least of the beholders position. To gain reasonable results for the project success examination, more mature projects are an ideal object of study, as it is assumed they abolish shortcomings and improve their processes over the time. Independent of their size (in LOC), in this study projects are considered as successful, which have a productive release version, are more than two years in the market, whose developer teams consist of more than five developers and which have a community above fifty participants. These conditions indicate a certain product maturity in conjunction with the market availability and show the ability to attract a large community. In this research, the term success describes the group of selected projects. However, this assumption does not reflect the actual project success.
These success selection criteria apply to one-fifth of the surveyed projects. These projects are object to the following analysis. It results that a higher motivation of the participants due to company needs (35.7%) exists. It can be concluded that the development of mature projects is often continued for commercial reasons. An increased release frequency, compared to the previous findings can be observed, as 39.2% releases once a quarter, while 32.1% release once half a year. The projects have increased testing activities in relation to the development time (40.7%) and three-quarter of the sample follow a structured testing approach. Users test on average 60.2% of the code. Moreover, 77.3% reported that user found hard bugs. Code reviews (84.5%), inspections (59%), walkthroughs (71.4%) and even peer review techniques (69.5%) are more frequently applied. An increased number of projects (55.9%) perform corrective actions before code commit and rework code or reject inappropriate code. It is remarkable that 42.8% of the projects introduce defect handling from project start, while 32.1% started this in the development phase. These projects track more frequently code defects (95.2%), documentation (63.1%), requirement (52.4%) and design issues (50.0%). Their defect reporting quality is noticeably high as three-quarter reported having useful bug descriptions as well as rules for defect classification and status tracking. In this sample, around half the projects have a knowledge transfer procedure. 84.5% of the respondents assess communication between user and developer as direct and efficient. Documentation has a high significance, for instance, 73.8% have process descriptions, 85.7% development documentation and 94.0% mention to have detailed product documentation or at least some drafts. The tool usage is conspicuously high, as more than 90% uses source code control tools, bug-tracking tools or mailing lists and roughly 60% apply instant messaging and test support tools. These projects have a slightly higher interest in the execution of QA practices (33.3%). However, an increased number of 42.8% report that through QA activities the adherence to standards and requirements is checked. Also 29.7% reply that QA triggers actions, resulting from interviews or inspections.

6. Conclusion

This research analyses quality assurance practices by surveying open source projects with a special focus on key practices in mature projects. The survey results provide evidence for the OSSD lifecycle described by Raymond [1]. Projects follow frequent releases, have a high user involvement and benefit largely from user testing and peer reviews. There is a significant growth of communities when projects mature and the existence of large user communities and developer groups could be observed. Developers mainly contribute due to personal or community needs, however a higher commercial motivation exists in large projects. In general, an excellent level of development knowledge can be observed. This leads to the assumption, that professionals know how to approach things 'right', which could explain the large number of successful projects collaborating in a loose manner sometimes with informal processes and fewer rules.

An investigation into more mature projects provides important insight into applied practices and may indicate reasons for the success of the OSSD model as shown as follows: These projects consider modularity of code already during design, which shows the concern to base the development on a solid architecture. Quality control activities before code commit have a higher importance. More time is spent on testing and testing approach is better structured. These projects efficiently leverage their community, benefiting from efficient user testing. Internal communication is rated as remarkable, which contributes positively to these processes. Defect handling processes seems better structured and include source code defects, requirements and documentation issues. Documentation has a higher significance. These projects emphasis widespread documentation and use this information to support the knowledge transfer to developers and users. Tool usage is high, making additional use of instant messaging, bug tracking tools and test support tools. The establishment of QA processes seems to play a more important role in larger projects and are underdeveloped in smaller projects.

The research findings contribute to an understanding of quality assurance practices and provide empirical evidence about applied processes, which OSS projects may use to improve their processes. The relations to project success criteria indicate some correlations between quality and succession but no causalities. Further research is required to explore QA practices and determine their relation to project success.

7. References

[1] E.S. Raymond, Linux and open-source success. *IEEE Software*, **16(1)**, 1999, pp. 85–89.

[2] J. Feller and B. Fitzgerald, A framework analysis of the open source software development paradigm. *in Proceedings of ICIS 2000*, 2000, pp. 58–69.

[3] F.P. Brooks, *The mythical man-month: essays on software engineering*. Addison-Wesley, 1995.

[4] G.G. Schulmeyer and J.I. McManus, *Handbook of Software Quality Assurance.* Prentice-Hall, 1999.

[5] R. Dunn, *Software Quality Concepts and Plans*. Prentice Hall, 1990, p. 11.

[6] L. Zhao and S. Elbaum, A survey on software quality related activities in open source. *ACM SIGSOFT Software Engineering Notes*, **25 (3)**, 2000, pp. 54–57.

[7] L. Zhao and S. Elbaum, Quality assurance under the open source development model. *The Journal of Systems and Software* **(66)**, 2003, pp. 65-75.

[8] T.J. Halloran and W.L. Scherlis, High Quality and Open Source Software Practices. *in Proceedings of ICSE 2002, Orlando, FL, USA, 2002*.

[9] G. Koru and J. Tian, Defect Handling in Medium and Large Open Source Projects. *IEEE Software,* **4**, 2004.

[10] M. Michlmayr, Software Process Maturity and the Success of Free Software Projects. *in* K. Zielinski and T. Szmuc (eds.) *Software engineering: evolution and emerging technologies*. IOS Press, 2005, pp. 3–14.

[11] P.R. Newsted, W. Chin, O. Ngwenyama and A. Lee, Resolved: Surveys have Outlived their Usefulness in IS Research. *in Proceedings of the 1996 International Conference on Information Systems, Cleveland, Ohio.*

[12] M.K. Malhotra and V. Grover, An assessment of survey research in POM: from constructs to theory. *Journal of Operations Management*, **16(4)**, 1998.

[13] K. Crowston, J. Howison, H. Annabi, Information systems success in free and open source software development: theory and measures. *Software Process: Improvement and Practice,* **11(2)**, 2006, pp. 123–148.

[14] D.W. Straub, D. Gefen and M.C. Boudreau, Validation Guidelines for IS Positivist Research. *Communications of the Association for Information Systems*, **14**, 2004.

[15] B. Massey, Why OSS folks think SE folks are clue-impaired. *in Proceedings of ICSE 2003*, 2003.

[16] C. Jensen and W. Scacchi, Modeling Recruitment and Role Migration Processes in OSSD Projects. Institute for Software Research, University of California, 2005.

[17] E.S. Raymond, *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary.* revised ed., O'Reilly, 2001.

[18] A. Porter, C. Yilmaz, A.M. Menon, A.S. Krishna, D.C. Schmidt and A. Gokhale, Techniques and processes for improving the quality and performance of open-source software. *Software Process: Improvement and Practice*, **11(2)**, 2006, pp. 163–176.

[19] X. Zhang and H. Pham, An analysis of factors affecting software reliability. *Journal of Systems and Software,* **50(1)**, 2000, pp. 43–56.

[20] L. Villa, Large free software projects and Bugzilla. *in Proceedings of the Linux Symposium, 23-26 July, Ottawa, Canada, 2003*

C.2Development of a Quality Assurance Framework for the Open Source Development Model

Tobias Otte
University of Wolverhampton
School of Computing and Information Technology
Wolverhampton, UK
Tobias.Otte@wlv.ac.uk

Robert Moreton
University of Wolverhampton
School of Computing and Information Technology
Wolverhampton, UK
R.Moreton@wlv.ac.uk

Heinz D. Knoell
Leuphana University of Lueneburg
Lueneburg, Germany
Knoell@uni.leuphana.de

Abstract

Under the Open Source Software Development (OSSD) model, many renowned products have been developed that appear to be high quality. Advocates of the OSSD model claim that the achievements of high quality largely come from user testing and peer reviews. We assume that Quality Assurance (QA) methods under the OSSD differ from traditional approaches and examined mid to large sized Open Source (OS) projects to identify applied practices. We found the existence of common key practices, but often processes are more informally applied without clear definitions. The findings contribute to the development of a QA framework, which suggests QA processes and product quality targets. The model offers processes to assure that the product achieves its quality goals. We conclude that the framework is an important element, which requires the balanced interaction of human factors, management skills and a suitable environment to achieve software quality.

1. Introduction

The basic idea behind the Open Source (OS) concept is simple and comprises the availability of the source code, the right to modify derived work, the free redistribution, the integrity of the author's source code and no restrictions for licenses [1]. Dietze [2] characterizes the Open Source Software Development (OSSD) model as a collaborative "bug-driven" development of globally distributed voluntary participants with high diversity of capabilities and qualifications of all actors. The interaction occurs exclusively through web-based technologies and development activities are executed in parallel, delivering frequent new software releases.
The OSSD model differs from traditional plan-driven approaches. While traditional methods have defined teams and requirements, the OSSD follows an iterative and parallel development approach with a user driven development direction, no central management, free participation, large development communities and effective user testing. Under the OSSD model quality management is affected by several aspects, such as the development methodology is often not documented, testing and Quality Assurance (QA) methods are informally applied, projects do not collate empirical evidence regarding quality and only few measurable quality goals are defined [3].
The OSSD model delivers successful products that seem to be high quality, such as Linux or the Apache Web Server. Hence, certain practices to assure software quality may exist. We hypothesize that successful projects follow similar key processes to assure software quality under the OSSD. The aim of this research is to investigate these key processes and to show their interactions in a process model. A qualitative research approach is selected and structured interviews with mature OSS projects are conducted to explore their QA practices. The findings contribute to the development of a QA process framework. Case study research is used to validate the research findings and show the correlation of QA key processes to project success.

2. Software quality assurance

Software Quality Assurance (SQA) comprises the planned and systematic activities to provide adequate confidence that an entity will fulfill requirements for quality [4]. The execution of SQA activities requires the introduction of a quality model to obtain process transparency, customer satisfaction, repeatable processes and methods. SQA involves both, process and product assurance [5]. Product-oriented approaches, such as the models by Boehm, Gilb or McCall aim to assure the quality of the product by evaluating their characteristics. Process-oriented approaches deal with the establishment of process definitions, the evaluation and improvement of software quality processes followed by the assumption that high quality development processes result in a high quality product [6]. Process-oriented approaches, which are completing

the ISO 9001, are for instance, CMM, CMMi, SPICE or Bootstrap. Process standards, such as ISO 12207, offer a wide range of management and engineering processes for the software development lifecycle. Product-oriented standards, such as ISO 9126 offer a multiplicity of measures to assure product quality targets.
Both approaches constitute the basis for our research towards the establishment of a QA model. The underlying approach is a process model, which triggers the assessment of product quality characteristics.

3. Recent research under the OSSD model

The examination of QA activities under the OSSD model has been subject to recent researches, such as by Zhao and Elbaum [7] [8], Halloran and Scherlis [9], Koru and Tian [10] or Michlmayr et al. [11]. These studies show evidence for the lifecycle claimed by Raymond [12]. Tool usage and user participation is high, which effectively supports testing. Defect handling is well structured, but documentation is often rare. The high quality in OSS relies on large communities, which enable effective debugging, user testing and suggestion of new features, high code modularity, frequent release cycles, application of peer reviews as well as an organized environment supported with tools [3].
In previous research, we surveyed mid to large size OSS projects and found that more mature projects do consider code modularity during software design [13]. These projects have stricter quality control activities before code commit and spend more time on testing which has a high efficiency. Defect handling processes are well structured and cover a broader range of topics. Documentation has a major importance and supports the knowledge transfer. Their internal communication seems remarkable, organizations are well defined and highly supported by tools. We observed that SQA processes are more often implemented in larger projects.
In recent years, the research in quality assurance activities under the OSSD model has increased. However, the collation of an omnibus QA process and measurement model, combining the processes and product view, is absent. The OSSD model may require the definition of a quality standard, which supports the entire lifecycle [14]. Our research aims to overcome this gap and to establish a tailorable QA framework for the OSSD.

4. Research method

The research adopts an anti-positivist epistemology approach using a social relativism paradigm [15]. A qualitative research approach is selected. First, the findings from the literature, survey and interviews regarding quality characteristics of the OSSD model are analyzed. Second, a generic model is developed and the model components are defined at attribute level to establish the fundamentals. Third, processes are consolidated into a QA framework and their characteristics are shown. Fourth, the measurement approach is examined and the applicability of product quality measures is discussed. Finally, the case study method is applied to validate the framework, using multiple methods of qualitative and quantitative techniques [16]. The combination of multiple methods enables triangulation, which lends greater support to the research conclusions [17]. The reasoning is drawn from the quantitative information resulting from interviews, documentation or archival records, which are supported by the quantitative findings from the questionnaire. The questionnaire results become more meaningful when correlated with critical qualitative information [16]. The case study method helps in hypothesis testing and requires multiple cases to confirm the existing theory but only a single critical case for rejection, because multiple cases yield more general research results [17]. The analysis is performed on the quantitative findings from the questionnaire to determine the correlation between applied processes and the project success using statistical methods. For ethical reasons all data are analyzed anonymously.

5. QA processes under the OSSD model

The research explores the development practices of various OSS projects in order to identify the key elements contributing to SQA. We observed that project requirements derive from different channels, such as direct feature requests by developers, user suggestions in mailings lists or from commercial partners. New feature requests are stored in public accessible lists or issue trackers to enable prioritization and status follow-up. The community or core team frequently conducts reviews and forces discussions to solve conflicting requests. The process gains an enormous importance in a distributed user-driven development approach. It sets the foundations for the design and subsequently for the development tasks and needs to reflect the evolving requirements during the entire lifecycle.
The documentation of development processes differs in its extent, however coding or style guidelines are frequently defined. Product documentation focuses on user oriented and technical oriented documentation, such as “API”. User documentation comprises for instance tutorials, guidelines or training materials and is often published using “Wiki” software. The establishment

of proper documentation has a high impact on quality, as its absence complicates the knowledge transfer to new participants [11].

The OSSD model contradicts Brooks [18] law that claims an increased complexity due to exponential growth of communication and cooperation with an increasing project size. We observed OSS projects are able to handle the complexity of large development communities, as these projects largely benefit from user testing and parallel development, which scales up with increasing team size. However, the establishment of an effective organization becomes a vital issue to produce high quality software. Projects need to emphasize the organizational setup to constitute a stable environment and enable efficient collaboration. Furthermore, the participation and attraction of volunteers affects software quality [11]. Appropriate structures are required to avoid frustration and keep people constantly motivated. This includes effective coordination of project tasks by the core development or management team, the implementation of issue tracking tools and the distribution of information via mailing lists. Efficient communication plays an important role in a geographically distributed development approach. It constitutes the basis for collaborative development processes and enables the frequent distribution of information. Instant messaging tools ease direct interactions and communication is mainly established via mailing lists or news groups, which enable an easier tracking of history.

Knowledge transfer processes gain an enormous importance to overcome a lack of know-how, as developers participate on a voluntary basis. We observed that some projects have strict rules and require mandatory documentation with code submission and often the information in the mailing list is used to capture knowledge. Appropriate process documentation eases the knowledge transfer for new developers and can be used to counteract the varying skill levels within the community. We found that knowledge transfer processes have a higher significance in larger projects. However, smaller projects organize these processes more informally and direct coaching by the lead developer is often done.

Michlmayr [11] indicates the importance of the infrastructure, which directly affects the project coordination and communication. The establishment of appropriate tools for the infrastructure has a significant impact on process quality as well as the collaboration. OSS projects frequently apply source code control tools or defect handling tools and closely integrate those predefined processes into their development approach. For instance, mailing lists or instant messaging enable collaborative development activities, such as pair programming. Thus, the adjustment and integration of tools offers the basis for managing the complexity within a collaborative, distributed development approach.

QA processes under the OSSD model focus on design, coding and testing. Villa [19] emphasized the need for an initial planning and a detailed documentation of design documents. An imperfect design phase which lacks of system analysis and the definition of the system architecture prior to coding might lead to severe quality issues when requirements evolve. Aberdour [3] noted that OSS projects neglect formal design and often go straight to programming. We observed that many projects conduct intensive design activities and establish a solid architecture such as the definition of an established class concept prior to coding. A few projects report informal design activities or establish design documents for documentation only and do not track changes. However, most projects made them publicly available and allow continual review changes by their community.

Projects vary in their adherence to coding standards and guidelines, as most of them continually review changes, perform corrective actions and reject inappropriate code. A few rely on peer reviewing by their community. Sometimes style checkers are used for format corrections. We observed that large projects control the quality of the code more strictly than others. Many perform intensive reviews, require mandatory unit tests before code submission or conduct automated testing.

Reviews and inspections processes assume a major importance in achieving software quality. These processes are conducted in parallel and scale up against system complexity [12]. The more people look at the code the higher is the ability to detect defects early. The application of reviews and inspections in the OSS becomes an integral element [20]. We found a high applicability of code reviews, inspections, walkthroughs and peer reviews techniques in OSS projects, as also emphasized by Raymond.

Larger OSS projects take advantage of their community size and benefit from user testing. We noticed that a majority of these projects reported high user testing efficiency, as the users found major defects or hard bugs. Michlmayr [11] noted that defect-handling processes could suffer from an imperfect bug-reporting quality or security critical updates are not available on short call. However, we observed mostly well structured defect handling processes, which can track common source code issues, while tracking of design, requirements or documentations issues are often neglected. Testing strategies vary in OSS projects. Some projects require unit tests with every committed code, while others use automated testing and set up a test suite with test cases. A few projects do approach testing informally. More than half of the observed projects

follow a structured testing approach. Often, integration or regression testing is performed before major releases.
Improper release management procedures and strategies negatively influence software quality [21]. The research has shown that a large number of OSS projects follow a feature based release strategy, scheduled in larger intervals. However a time-based approach is often selected when closer to release. Quality is obtained by code freezes and intensified integration or regression testing towards the delivery of release candidates.
OSS projects apply the definition of quality targets and their assurance is mostly informal. We noticed the existence of minor goals, such as some benchmarks or code coverage targets. More surprising is the fact that no strict quality management approach could be observed with the aim of defining quality goals and their assurance. The inference can be made that OSS projects neglect quality management processes or that they do not make a clear distinction of software quality assurance processes. The definition and applicability of QA processes across OSS projects is not prevalent. We identified a higher significance of QA processes in larger projects. SQA processes are more frequently established when projects evolve in size and face an increasing complexity of their development processes. Some larger projects define dedicated responsibilities to achieve the execution of SQA tasks. Often projects perform SQA tasks more informally without any strategy or planning. The analysis of the applied QA practices indicates the existence of continuous process improvement approaches in OSSD projects. However, none of the projects explicitly named this process and its execution. It can be concluded that the spirit of continuous improvement implicitly exists in the community and developers are empowered to express their ideas for improvement.
In summary, we suggest the following key processes to support QA practices under the OSSD: Product and Process Documentation, Coordination and Team Communication, Infrastructure and Tools, Knowledge Transfer, Requirements Evaluation and Design, Software Engineering Control, Peer Reviews and Inspections, Verification and Validation, Release Management, Quality Management, Software Quality Assurance and Continuous Process Improvement.

6. Quality assurance framework

The Quality Assurance Framework for Open Source Software (QAfOSS) consolidates a process model and a product assessment approach. The QA process model describes perspectives, orientation and deliverables of the processes and methods. An underlying process framework with a subsequent process structure constitutes the basis for the QA process model. First, the taxonomy of the process model is defined. A generic model with a hierarchical process structure is developed, describing process targets, outcomes and activities. Second, the framework is collated and the interactions are shown.

6.1. Generic process framework

The generic process framework, as depicted in figure 1, defines the taxonomy of the process model and considers process objectives, characteristics and evaluation criteria. Wang and King [22] define the process domain as a set of ranges of functional coverage and suggest a differentiation into organization, development and management. In addition, the domain resource is introduced to show a detailed partition of organizational and infrastructural aspects. The process framework distinguishes the following process objectives:

- Processes focusing on the management
- Processes with an administrative purpose
- Processes affecting directly the resources
- Processes with focus on the development lifecycle and on the product

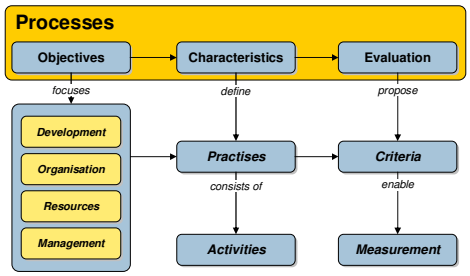

Figure 1. Generic process framework

Each domain consists of process groups, which cluster specific processes. The taxonomy of processes follows a hierarchically structure and comprise process groups, features and practices. The highest node represents the process group. A process is a set of sequential practices, which are functionally coherent and reusable for software project organization, implementation and management [22]. Each process is described by its purpose, outcomes and practices. The practices describe activities that contribute to the process fulfillment and constitute the lowest level of the model. The evaluation of the practices requires the definition of measurement criteria. These criteria allow assessment of the implementation of the practices within the development lifecycle.

Table 1. QA process framework

Management
Requirement Management
Requirements Capturing
Continuous capturing, documentation, tracking, tool support
Requirements Review
Recurring reviews, public tracking, adjust planning
Documentation Management
Process Documentation
Documented processes, dev. style and coding guidelines
Product Documentation
Updated technical and user documentation., online accessibility
Release Management
Release Strategy
Define strategy, scope management, ensure stability
Build and Release Check
Quality control, code freeze, release candidates, testing
Quality Management
Define quality targets, set standards, measure product quality
Software Quality Assurance
QA strategy, check adherence to standards, perform reviews, identify deviations from plan, reporting to team
Organization
Coordination and Team Communication
Project Organization
Define org. structures, role model, authorizations, guidelines
Project Coordination
Information accessibility, collaborative development rules
Team Communication
Establish communication tools, guidelines, review efficiency
Continuous Process Improvement
Process Change Management
Improvement targets, establish process community wide
Defect Prevention
Identify common causes, systematically eliminated
Resources
Knowledge Transfer
Knowledge Capturing
Force document. before commit, public archives, reviews
Team Education
Skill level review, team coaching, how-to guidelines
Infrastructure and Tools
Collaborative tools, high integration, documented guidelines
Development
Software Engineering
Design Control
Design reviews, quality targets, define test requirements
Development Control
Adherence to standards and quality, force documentation
Continuously Code Quality Control
Quality control before submit, versioning, rework, rejection of inappropriate source code, evaluate risks and side effects
Review and Inspections
Walkthroughs, inspections, peer reviews, corrective actions
Verification and Validation
Defect Management
Defect management, classification, risks, control, tracking
Unit, Integration and Regression Testing
Test strategy, planning, test cases, consider varying product configuration or system architecture, tool support, tracking

6.2. Process model

The process model describes QA processes independently of a development method. Organizations can adapt and tailor them to their needs. A set of common QA processes are defined in relation to the definition of a reference model, which reflects the benchmarked best practices in software development [22].
The origins of our research are international standards, such as ISO 12207 or CMM. We used the findings from literature and interviews to identify best practices of QA processes in the OSSD. The developed process model comprises the main QA activities as summarized in table 1. The model describes relevant QA process groups with subsequent processes and indicates sample practices.

6.3. Product quality

Well known quality models, which define a set of measures associated with software metrics, have been introduced by Boehm et al. [23] and McCall et al. [24]. The ISO 9126 standard consolidates these views and offers a quality standard, consisting of internal, external and "quality in use" criteria. The standard comprises a multitude of measures and needs to be tailored to project specific requirements. ISO 9126 is applicable for software development under the OSSD. However, more attention needs to be given to measure affecting accessibility, reusability, modularity and code quality. The accessibility of the source code is a key element of the development approach and influences its changeability. It is precondition for user testing, inspection and peer reviews, which affects testing effectiveness and reporting quality. A high modularity presupposes a unitized specification, impacts reusability, maintainability and lowers the defect density [3] [14]. A high importance during the development needs to be given to code quality criteria, such as understandability, completeness, conciseness, portability, consistency, maintainability, testability, usability, reliability, degree of structured or efficiency.
ISO 9126 offers a broad range of metrics and requires a selection and adjustments by each project according to their development requirements and targets. The definition of project specific measures is mandatory to execute quality management.

6.4. Process measurement approach

A process capability measurement method is introduced to assess the level of process implementation within an organization. The method determines a capability rating for an organization and assesses each process against product quality characteristics [25] such as functionality, usabil-

ity, reliability and efficiency. The functionality assesses the process compliance, its completeness and suitability. The usability mainly focuses on the understandability, learnability and operability aspects of a process. The reliability is defined by the process fault tolerance and its maturity, while the efficiency rates the process performance in terms of time behavior or the resource utilization. Thus, the measurement model delivers the nearest mapping of the framework to the actual processes on an aggregated level. A questionnaire was developed to evaluate applied processes, which is available online www.qafoss.org/data/qafoss_model.html.
Each existing process is evaluated with eight Process Capability Scores (PCS), while maximum eight additional PCSs can be obtained for the evaluated process characteristics, such as functionality (-4 to +4), usability (-1 to +1), reliability (-1 to +1) and efficiency (-2 to +2). "Under fulfillment" is penalized with negative scores, while "over fulfillment" is rewarded. For instance, a poor process may be downgraded to zero. In total, 23 processes are assessed, which gives a maximum of 368 PCSs. An individual importance factor is used to weight the results.
In addition, an individual Project Success Score (PSS) is determined to evaluate the correlation of the process capability findings. The respondents were asked to estimate a success factor on a scale from 1 to 10, where 10 means high success. The success evaluation comprises the system and information quality, usability aspects as well as the overall user satisfaction [26]. For instance, a 10 would describe a mature project, with high attraction, high popularity, clear processes and outstanding deliverables.

7. Framework validation

The framework was validated using case study research, exploring different organizations. The site selection considers "literal" and "theoretical" replication. The "literal" replication is used where similar results may be predicted, while "theoretical" replication is applied where contradictory results may occur [27]. Different projects were selected, based on the degree of fulfillment of the OSSD model, the complexity, the maturity level, the development approach and the diversity of application. Structured interviews were used to collect data about applied processes, development approach and methodology, in order to classify the organizations to their type of replication.
The aim of the case study research is to show the process compliance of the QAfOSS framework and to explore its correlation to project success. Seven case studies have been conducted with different application types, such as Software Development, Internet, ERP/ CRM/ Financial or Office/Business. The first six cases focus on a literal replication, while the last theoretical case examines a less successful project. Within the case studies, qualitative data is collected and a process capability measurement is derived. Purely described data is converted into quantitative data to perform statistical analysis [16], using SPSS.

7.1. Literal case studies

The literal case studies distinguish different types of application, with mid- to high level of complexity and different type of motivation, such as community or commercially driven projects. The studies show, that high commercial interests influence management, organization and resources. For instance, paid full-time resources often take over a professional management role and offer a huge capacity increase.
Some commercially driven projects follow a hybrid approach, combining traditional and OSS development model. An observed strength of those projects is their strict management focus on schedule, planning, steering and coordination of tasks, triggering actively the OSS community. Their project organization follows mostly a role model, with different access rights to the repository. In general, the project communication is well established, supported by public and private mailing lists, use groups and instant messaging. Some projects prefer use groups that enable an easier tracking of historical information compared to mailing lists.
All projects manage to benefit from an experienced team and a direct communication within their community, which enables adjustments of the development direction and leads to the improvement of their product quality, due to efficient testing and debugging feedback.
Mostly a central requirement management is done, using an issue tracking tool or the project documentation. The integration of requirements follows defined processes, such as formal reviews by the core development team or the community, before new feature requests are chosen to be integrated into the design.
The documentation of processes commonly focuses on style and coding guidelines and mostly complete product documentation concerning user- and technical documentation is available. Projects often face problems with poor quality of the user documentation, such as lack of date-stamping and lack of tools to check the documentation quality.
Typically knowledge capturing concentrates on mandatory documentation alongside the contributed source code. While many projects require the contribution of only highly experienced developers, some projects monitor developer skill levels and coach them accordingly.
All projects leverage tools which are highly tailored to their needs and support their devel-

opment processes. With increasing project complexity, guidelines and process descriptions are mostly available. The analysis of the engineering processes shows that a comprehensive system design is part of the development. Results are documented, but sometimes projects fail to keep the documentation up-to date or are lacking design control.
The main strengths are continuous code quality checks, such as reviews, inspections, walkthroughs and peer reviews that are conducted by their communities. Beside manual checks, automated code style checkers are used to control the development activities. The applicability of pair programming faces problems in a distributed environment with different time zones. However, quality controls before commit or mandatory unit tests contribute to high code quality. Some projects perform integration tests prior to a new release, while some commercial driven projects execute, in addition, a complete regression test or conduct automated tests against design documents.
A key success factor is the enabling of the community to perform efficient testing or debugging, due to well-structured defect handling with classification and prioritization. Furthermore, the achievement of a high reporting quality and defined testing processes, using a full testing suite are essential components. Some large projects assign responsibilities to a dedicated QA team for risk based testing, the approval of new releases and the coordination of defect fixing prior to a release. New releases are strictly managed, mostly following a feature-based approach, while often a time-based approach is used when it comes closer to a release. Typically, a feature freeze is conducted and release candidates are used for community testing. However, the management of quality targets is mostly neglected. Only some projects consider code coverage analysis or focus on performance profiles or benchmarking. Quality targets, SQA activities as well as continuous process improvement approaches are informally managed, but there is a strong motivation for defect prevention. In summary, the literal case studies show a high compliance with the process model, with minor deviations regarding quality management, process improvements and knowledge transfer processes.

7.2. Theoretical case study

The theoretical case study analyses an OSS development project with a mid complexity. Unlike previous studies the project has a low maturity level, is driven by part-time volunteers and has a mid-to-low activity level. Project processes are lacking due to several issues, such as organizational setup or management. There is no strict management approach to steer the project and to ensure a suitable environment. Requirements management processes are mostly neglected and only a simple capturing in to-do-lists or the wiki documentation exist, but transparency about required features and their status is lacking. The community performs requirement reviews informally. The documentation comprises coding style guidelines or testing guidelines that are published on the project wiki. However, the documentation is messy, outdated, scattered and lacking in structure. The organizational setup follows the classical OSS roles model with a flat structure and restricted access rights to the repository. The coordination of tasks occurs via the rudimentary bug lists, with limited functionality. The main communication is established using the mailing list, in order to spread and to capture information. Any supporting documentation for joining developers does not exist, except loosely coaching by experienced developers. The project uses an OSS tool based infrastructure with imperfect functionalities. The work environment is underdeveloped. For instance, defect handling tools are used that offer insufficient control for tracking and coordination. This has a negative influence on the efficiency of testing processes, because reporting quality or timely feedback to community members is missing. Furthermore, the defect handling process is not well maintained and some defects remain in situ for years. The quality control of engineering tasks is informally done. Nevertheless, code reviews and inspections are continuously performed, as soon as changes are posted via the mailing list. Testing activities are addressed to the community, but a strategy, planning or documentation is not observable. The release management strategy follows a time-base approach with defined targets but an outdated schedule. Meanwhile, a large discrepancy between planning and actual release version exists and no maintenance or adjustments of the plan are done. The project follows the idea of release candidates towards the development of a stable version. Quality management targets and SQA tasks are lacking and not documented. The philosophy for continuous process improvement exists, but there is an unwillingness on the part of management to force such improvements. A lack of management skills is seen as the central issue; no sufficient development environment is established, the infrastructure is lacking and there is a lack of focus on central tasks, such as organization, documentation and the triggering of SQA activities. Furthermore, the project is short on resources, due to part-time contributors with time constraints.
In conclusion, the project cannot leverage its community efficiently, as its environmental processes are not transparent and risks the loss of ‘attraction’.

Table 2. Case Study Results

Process Capability	CASE 1	CASE 2	CASE 3	CASE 4	CASE 5	CASE 6	CASE 7
PCS total capability score	226	272	248	246	255	233	162
PCS_C % coverage	61.413%	73.913%	67.391%	66.848%	69.293%	63.315%	44.022%
PCS_CW % coverage weighted	**68.064%**	**75.860%**	**73.109%**	**73.529%**	**69.988%**	**67.951%**	**53.547%**
Success Measurement							
PSS estimated	**8.000**	**7.500**	**8.500**	**8.000**	**7.000**	**7.500**	**5.000**

7.3. Case study analysis

The qualitative findings are supported by a quantitative analysis, using the process capability measurement approach. The findings are depicted in the two-dimensional radar chart, showing a percental value for the assessed capability of the rated processes. The case study seven shows a low process compliance with the QAfOSS framework. Especially management, organizational and resource related activities are beyond the process targets, as shown in figure 2.

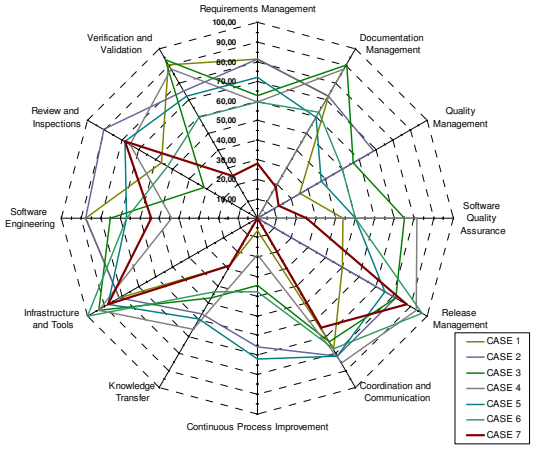

Figure 2. Case study process determination results

The quantitative information comprises the determination of the process capability, based on the measurement model and the project success factor, which reflects the primary quality targets [26]. The process capability findings face several limitations, which might have an influence on the results, such as a differing objectivity of the respondents or the usage of different process assessment standards.

Table 2 summarizes the quantitative findings, showing the total PCS, the percentage model coverage (PCS_CW) according to weighted score and the estimated PSS value. The PCS score is weighted with an individual importance factor for each process to emphasize the key processes and to amplify the percentage model coverage. The group of successful projects has high process coverage, while the less successful project shows significant deficits in management, organizational and engineering, resulting in a low PCS. The statistical analysis shows a significant high positive correlation between the PCS_CW and the estimated PSS (r = .867, p = .006).

Thus, successful projects mainly follow the key processes of the QAfOSS model. The results may provide evidence for the validity of the model and its process capability measurement approach. However, the findings do not imply that purely implemented processes result in project success. In addition to the establishment of QA processes, project success depends on the integration of several aspects, including management, organization or social considerations.

8. Conclusion

The research examines QA practices in mid- to large OSS projects and suggests a QA framework. Within the case study research, the model is validated and shows a high correlation with applied processes in successful projects. The framework constitutes as a tailorable reference model and consolidates important key processes that contribute to SQA:

Requirement management needs to reflect evolving feature requests and sets the foundation for development, testing and acceptance of the product. Documentation management affects the knowledge transfer and is key factor to share, enrich and capture know how within the projects. Strict project coordination is required to force the management of activities to adhere the collaboration. Communication processes and a suitable infrastructure with integrated tools are a key factor to enable the distributed development approach. A project requires a certain degree of ‘attraction’ to attain the critical mass of voluntary contributors to utilize the advantages of large teams. The development processes ought to consider an initial planning and design activities to avoid quality issues when the requirements evolve. Quality control processes to check the adherence to standards and the use of reviews become a major importance to assure quality. A project needs to set up reasonable testing processes and defect management to leverage their community size effectively. Release management becomes an integral element for QA. It comprises scope management, triggers testing activities and forces the establishment of release candidates to assure a widely verification. Quality management is needed to define, measure and control specific quality targets. SQA processes obtain a central role to

achieve the implementation and the effective interaction of the QA framework elements.
We assume that the quality of the product depends on the key processes used to develop it. However, the assumption is misleading, that the proper implementation of the processes, which produced in the past high quality, will lead to successful high quality products [28]. The applicability of the model does not guarantee high software quality, but the implementation of the processes provides adequate quality assurance. QA is highly influenced due to user testing and debugging, therefore projects need to concentrate on enabling tasks, such as structured organization, good documentation and effective communication. However, the achievement of a high product quality depends on several factors, such the effective interaction of people, management, environment, development approach and the degree of SQA measures.
An in-depth study of the QA process model is required to prove the benefits. Further research may explore the dependencies of the key elements that contribute to software quality under the OSSD.

9. References

[1] OSI, *The Open Source Definition* [online]. Open Source Initiative, [cited 18th March 2008]. <http://www.opensource.org/docs/definition.php >.

[2] S. Dietze, Agile Requirements Definition for Software Improvement and Maintenance in Open Source Software Development. *in Proceedings of SREP'05, 2005.*

[3] M. Aberdour, Achieving Quality in Open Source Software. *IEEE Software*, **24(1)**, 2007, pp. 58–64.

[4] ISO/IEC 12207, Systems and Software Engineering – Software life cycle processes. ISO, Geneva, 2007, p.6.

[5] R. Fairley, *Software Quality Engineering Course Notes*. 16. April 1997.

[6] B. Wong, The Different Views of Software Quality. *in* E.W. Duggan and H. Reichgelt (eds.) *Measuring Information Systems Delivery Quality*. Hershey: Idea Group Publishing, 2006, pp. 55–88.

[7] L. Zhao and S. Elbaum, A survey on software quality related activities in open source. *ACM SIGSOFT Software Engineering Notes*, **25(3)**, 2000, pp. 54–57.

[8] L. Zhao and S. Elbaum, Quality assurance under the open source development model. *The Journal of Systems and Software* **(66)**, 2003, pp. 65-75.

[9] T.J. Halloran and W.L. Scherlis, High Quality and Open Source Software Practices. *in Proceedings* ICSE 2002, *Orlando, FL, USA*, 2002.

[10] G. Koru and J. Tian, Defect Handling in Medium and Large Open Source Projects. *IEEE Software,* **4**, 2004.

[11] M. Michlmayr, F. Hunt and D. Probert, Quality Practices and Problems in Free Software Projects. *in Proceedings of the First International Conference on Open Source Software Systems, 11-15 July 2005, Genova*, pp. 24–28.

[12] E.S. Raymond, *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary.* revised ed., O'Reilly, 2001.

[13] T. Otte, R. Moreton and H.D. Knoell, Applied Quality Assurance Methods under the Open Source Development Model. *to appear in Annual IEEE International Computer Software and Application Conference 2008.*

[14] I. Stamelos, L. Angelis, A. Oikonomou and G.L. Bleris, Code quality analysis in open source software development. *Information Systems Journal*, **12(1)**, 2002, p. 58

[15] J. Iivari, R. Hirschheim and H.K. Klein, A Paradigmatic Analysis Contrasting Information Systems Development Approaches and Methodologies. *Information Systems Research*, **9(2)**, 1998, pp. 164–193.

[16] S. Lubbe, The Development of a Case Study Methodology in the Information Technology (IT) Field: A Step by Step Approach. *Ubiquity - An ACM IT Magazine and Forum*, **4(27)**, 2003, pp. 26.

[17] I. Benbasat, D.K. Goldstein and M. Mead, The Case Research Strategy in Studies of Information Systems. *MIS Quarterly*, **11(3)**, 1987, pp. 369–386.

[18] F.P. Brooks, *The mythical man-month: essays on software engineering*. Addison-Wesley, 1995.

[19] L. Villa, Large free software projects and Bugzilla. *in Proceedings of the Linux Symposium, 23-26 July 2003, Ottawa, Canada.*

[20] J. Dinkelacker, P.K. Garg, R. Miller and D. Nelson, Progressive Open Source. *in Proceedings of ICSE'02,* Orlando: ACM Press, 2002, pp. 177–184.

[21] A. Porter, C. Yilmaz, A.M. Menon, A.S. Krishna, D.C. Schmidt and A. Gokhale, Techniques and processes for improving the quality and performance of open-source software. *Software Process: Improvement and Practice*, **11(2)**, 2006, pp. 163–176.

[22] Y. Wang and G. King, *Software Engineering Processes – Principles and Applications*. Boca Raton London New York: CRC Press, 2000, pp. 50–52.

[23] B.W. Boehm, J. R. Brown, H. Kasper, M. Lipow, G. Macleod and R. Merritt, *Characteristics of software quality*. TRW series of software technology, 1978.

[24] J.A. McCall, P.K. Richards and F.G. Walters. *Factors in software quality*. RADC-TR-77-369, 1977.

[25] M. Satpathy, R. Harrison, C. Snook and M. Butler, A Generic Model for Assessing Process Quality. *in Proceedings of the IWSM2000, 2000.*

[26] W.H. Delone and E.R. Mclean, Information Systems Success: The Quest for the Dependent Variable, *Information Systems Research*, **3(1)**, 1992, pp. 60–95.

[27] R.K. Yin, Case Study Research, Design and Methods. 3rd ed., Sage Publications, 2002.

[28] E.R. Baker and M.J. Fisher, Software Quality Program Organisation. *in* G.G Schulmeyer and J.I. McManus